A Method for Aiding Requirements Analysts in Requirements Elicitation for Large Software Systems

Leah Goldin

A Method for Aiding Requirements Analysts in Requirements Elicitation for Large Software Systems

Research Thesis

Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Science

Leah Goldin

Submitted to the Senate of the Technion — Israel Institute of Technology
Tammuz 5755 HAIFA July 1994

This research was carried out in the Faculty of Computer Science under the supervision of Professor Daniel M. Berry.

I would like to thank Dan Berry for being teacher, colleague, and friend.

I wish to thank the members of my thesis committee, Amiram Yehudai, Janos Makowsky, Yona Lavi, and Yossi Gil. Many thanks for their insightful suggestions on the writing of this thesis.

Eliezer Kantorowitz, Michael Rodeh, Amiram Yehudai, and Janos Makowsky provided helpful comments at the decisive point of the thesis proposal.

Specially, I would like to acknowledge the assistance of Yael Dubinsky and Haim Roman with operating system problems, and Chava Shamir for her constant support and encouragement.

For Simcha
Ayelet
Chemi
Hadar
Zur

Table of Contents

page

1 Introduction 2
1.1 The Problem 2
1.2 Requirements Engineering 5
1.3 Requirements Elicitation 6
1.4 Existing Requirements Engineering Methods, Tools, and Systems 8
1.4.1 Tools and Systems for Requirements Analysis 8
1.4.2 Deficiency of Past Work for Requirements Elicitation 11
1.4.3 Requirements Elicitation Methods and Tools 12
1.4.3.1 Natural Language Processing 12
1.4.3.2 Social Perspectives 13
1.4.3.3 Guiding Systems 14
1.5 Envisioned Requirements Gathering Environment 15
1.6 Plan of Thesis 17
2 Abstraction Identification 18
2.1 Operational Definition and Assumptions 18
2.2 Existing Abstraction Identification Tools 20
2.2.1 Grammatical Parsers 20
2.2.2 Repeated Phrase Finder 21
2.2.3 Lexical Affinities 22
2.2.4 Remaining Weaknesses 24
2.3 New Approach 24
2.3.1 Motivation 24
2.3.2 Formal Description 25
2.3.3 How the New Approach Avoids Weaknesses of Previous Approaches 30
2.3.4 Possible New Weaknesses of the New Approach 31
2.3.5 AbstFinder Program 33
2.3.6 Performance Analysis of Program 36
2.3.6.1 Value of *WordThreshold* 36
2.3.6.2 Complexity of Current Implementation 37
2.3.6.2.1 Time Complexity of Prototype 37
2.3.6.2.2 Space Complexity of Prototype 37
2.3.6.3 Alternative Implementations 38
2.3.6.3.1 Complexity of Alternative Implementations 39
2.3.6.3.2 Other Alternatives 40
2.3.6.4 Final Complexity Assessment 40
3 Scenarios for Usage of AbstFinder 41
3.1 Learning What Words to Ignore 41
3.2 What to Do with a Well-Organized Document 41
3.3 Zooming 42
3.4 Iteration to Final List of Abstractions 44
3.5 Combinations of Scenarios 47

Table of Contents Contd.

4 Evaluation of AbstFinder 48
4.1 How to Evaluate a New Method or a Tool 48
4.1.1 The Evaluation Plan 51
4.2 Case Studies 52
4.2.1 Findphrases Case Study. 52
4.2.2 Flinger Missile Case Study 53
4.2.3 RFP Case Study 55
4.2.3.1 findphrases vs. AbstFinder 62
4.2.4 Results 63
4.3 Indexing vs. Abstraction Finding 64
5 Abstraction Organization Exploration 67
5.1 What is an Abstraction Network 67
5.1.1 Using the Abstraction Network—The Perspective of the Elicitor 70
5.2 Problem Exploring with Hypertext 73
5.2.1 Hypertext-Related Work 73
5.2.2 Generate Abstraction Network On Top of Hypertext 75
5.2.3 An Abstraction Network Case Study 76
5.2.3.1 Findphrases Case Study 77
5.3 Abstraction Network Exploration Summary 78
6 Conclusions 80
Bibliography 82
Appendix A AbstFinder Implementation Considerations 90
A.1 AbstFinder Output 90
A.2 How does the Elicitor use AbstFinder Output 91
A.3 AbstFinder Input Filtering 92
Appendix B Manual Pages of AbstFinder Tools 94
Appendix C Results and Data of the findphrases Case Study 108
C.1 Source Transcript of the findphrases Case Study 108
C.2 Ignored Files Used in the findphrases Case Study 111
C.3 AbstFinder Results in the findphrases Case Study 112
C.4 Decomposition of the findphrases Case Study 127
Appendix D Results and Data of the Flinger Missile Case Study 128
D.1 Source Transcript of the Flinger Missile Case Study 128
D.2 Ignored Files Used in the Flinger Missile Case Study 136
D.3 AbstFinder Results of the Flinger Missile Case Study 137
D.4 The Flinger Missile Transcript after Straining 143
Appendix E Results and Data of the RFP Case Study 150
E.1 Source Transcript of the RFP Case Study 150
E.2 Ignored Files used in the RFP Case Study 156
E.3 AbstFinder Results on the RFP Case Study: First Iteration 158
E.4 AbstFinder Results on the RFP Case Study: Last Iteration 167
E.5 ASSR - Allocated System Software Requirements for the RFP 169
E.6 Sub-abstractions of Testing Abstraction (Summary) 172
E.7 Sub-abstractions of Safety and Fail Abstractions 175
E.8 Results of findphrases on the RFP Case Study 178

Table of Contents Contd.

Appendix F Using Abstraction Network of Findphrases Case Study 186
F.1 Makefile for generating links automatically 191
F.2 Example of using Hyperties with logging (findphrases) 195
F.3 Hyperties input file with mark-up (findphrases) 198

List of Figures and Tables

page

Figure 1: Network of Nodes 16

Figure 2: Links 16

Figure 3: Comparing shorter sentence to circular shifts of the longer 29

Figure 4: Comparing doubled longer sentence to shifts of the shorter 30

Figure 5: First part of AbstFinder output 34

Figure 6: Second part of AbstFinder output 35

Figure 7: Finding sub-abstractions by zooming 43

Figure 8: Iterative abstraction identification 46

Table 1: The abstractions of findphrases 53

Figure 9: An illustration of the RFP case study 56

Table 2: The Case Studies Summarizing Criteria 64

Table 3: Case Studies comparison 65

Figure 10: The abstraction network 69

Figure 11: A segment of possible IBIS-style discussion network 71

Figure 12: The hypertext concept 74

Figure 13: AbstFinder Input/Output Architecture 93

Figure 14: Decomposition of findphrases by Aguilera 127

Table 4: The links generated automatically by AbstFinder. 187

Figure 15: Hyperties with findphrases 188

Figure 15: Hyperties with findphrases (Cont.) 189

Figure 15: Hyperties with findphrases (Cont.) 190

Abstract

The importance of obtaining good requirements for software-based systems cannot be understated. Many methods and tools have been devised for analyzing requirements, but only recently has there been a focus on what needs to be done before the analysis can begin, that is, eliciting the information from the client and identifying the abstractions contained in this information. As a step towards solving the problems of requirements elicitation, this work motivates and describes a new approach for a tool, based on traditional signal processing methods, to help find abstractions in natural language text, such as one might receive from the client of a proposed software system.

In order to design the tool, it was necessary to determine effective ways of identifying abstractions in natural languages transcripts of client interviews. The tool, AbstFinder, was implemented and a number of case studies demonstrated the effectiveness of AbstFinder even on industrial-sized examples.

The key measures of the effectiveness of AbstFinder are: (1) its coverage, and (2) how summarizing it is. An RA will not be willing to be assisted by any tool unless he or she is confident that it is covering. However, presenting all the input does not help the RA either. To prevent from overwhelming the RA, the tool should reduce what he or she has to examine. In any case, the human RA still has to do the *thinking* with the output of the tool; consequently it is not necessary that the tool exhibit any intelligence, especially if the intelligence might cost some of its coverage. The case studies have shown that AbstFinder is indeed covering and summarizing.

With any scheme of automated assistance, scenarios for usage should be defined. Typically, in the process of abstraction identification, even the most intelligent elicitor cannot abstract a full transcript all at once. Thus, different scenarios for using AbstFinder were identified, and these include the following activities: learning what words to ignore, zooming, and iteration to a final list of abstractions.

Once the abstractions are recognized, they have to be organized so that all the text dealing with any abstraction can easily be found. Only then, the elicitor will be able to finish the elicitation, i.e., filtering out inconsistencies, identifying absence of information, negotiating with the clients and users, and updating the abstraction content if necessary.

The work also explores the use of hypertext as a medium in which to organize the abstractions that have been identified. Since there is a lot of work being carried out by others in the problems of usability of hypertext itself, e.g., how to avoid getting lost in hyperspace, the present exploration was limited to identifying needed functionality and made no attempt to evaluate usability issues. There has also been a lot of research recently in the use of hypertext in software engineering in general. Thus, the present exploration is intended to bridge abstraction identification to the other work.

1 Introduction

1.1 The Problem

Requirements more often are ill-defined, fuzzy, incomplete, or simply incorrect with respect to users' needs. Problems in the system caused by deficiencies in software requirements are often not identified until well after the system is deployed, or are thought to be caused by bad design or limitations of computing technology.

It is well known that as much as 60% of the errors that show up during a system's life have their origin in the requirements gathering and specification stage [Dav90, Sch92]. It is also well known that the cost to correct an error found in the development stages of system development is orders of magnitude higher than to correct the same error found during the requirements gathering and specification stages [Boe81]. The importance of getting the requirements right cannot be underestimated.

On the other hand, it appears that the least understood step of systems development is the requirements gathering and specification stage, and that within this stage, gathering is less understood than specification.

The problem is that there is a tremendous gap between the client's needs and the software engineer's understanding of the client's needs. The gap is widened by the fact that the client may not even be able to verbalize his or her own needs. The client speaks with fuzzy sentences replete with tacit assumptions, and the software designers are just not able to identify his or her intentions.

Furthermore, requirements are subjected to continual change, and almost always this change impacts the software. This fact has affected the life-cycle models in which these risks are considered, e.g., as in Boehm's spiral model [Boe88].

The Software Engineering Institute (SEI) was asked by the Air Force Systems to perform a near-term study assessing the nation's capacity to produce software for the Department of Defence (DoD). The SEI found [SSKLW90] that the executives' perspectives on the relative importance of factors that contribute to the failure of military system development contracts to meet schedules and costs are (The numbers are in a scale of 1 through 5 with 5 being "most important".):

1. inadequate requirement specification (4.5),
2. changes in requirements (4.3),
3. shortage of system engineers (4.2),
4. shortage of software managers (4.1),
5. shortage of qualified project managers (4.1),

6. fixed-price contracts (3.8),

7. inadequate communication for systems integration (3.8),

8. insufficient experience as team (3.8),

9. shortage of application domain experts (3.6),

10. integration of contractor/subcontractor efforts (3.5),

11. new application domain (3.5),

12. inadequate software development tools (3.4),

13. turnover of personnel (3.3),

14. shortage of skilled programmers (3.2),

15. complex hardware (2.8), and

16. shortage of electrical engineers (2.7).

Of these, 1, 2, 9, and 11 are requirements related. Note that shortage of application domain experts, the ones that have the knowledge about the tacit assumptions of the problem domain, makes it almost impossible to get the needed requirements information.

Many system design or programming methods, e.g., those of Jackson [Jac75], Parnas [Parnas 1972], Booch [Boo86], Myers [Mye79], Orr [Orr77, Orr81], etc., start from an assumed clear statement of requirements and show how to arrive at a design of a program meeting those requirements. However, none of these methods really explain how these requirements are obtained in the first place. It is clear that writing of the requirements is a *major* part of the problem solution, and that when this writing is done properly, many pitfalls in the path of delivering the required system may be avoided.

Large E type [Lehman 1980] software, for which it is difficult or even impossible to obtain clear requirements, is usually developed for a client organization in which there are many people who have some view or say as to what the desired system should do. These views range from being deceptively similar to each other through being totally unrelated to each other to being totally inconsistent with each other. It is no wonder that the distillation of these views into a consistent, complete, and unambiguous statement of the requirements, albeit in natural language, is a *major* part of the problem of developing software which meets the client's needs. Therefore, it is essential to have methods and tools that help in distilling these many views into coherent requirements.

A basic reality of large E type systems is that they are too large for one person to keep in mind. Consequently, both the client and the producers of such systems are required to be teams of people working together. Therefore, it is necessary that the tools support cooperative [CSCW88] work by the members of these teams.

An experiment that was done by Martin and Tsai [MT88, SMT92] is very telling. The goal of the experiment was to study the earliest phases of software development by cataloguing the number, types, and severities of errors detected in the user requirements. Ten teams worked in isolation using different methods, each developing from the same user requirements. Each team was organized as an independent working group with four software engineers, one of whom was the controller who was responsible for scheduling tasks and collecting experiment data. The subject of the user requirements was a centralized railroad traffic control (CTC). Each team detected user requirement errors, assessed feasibility, and tried to arrive at an informal understanding of the user's needs.

The user requirements were only about 10 pages long, and the user believed that only one or two mistakes would be found there. But the results of error identification by the ten groups were amazing. A total of 92 requirement errors were identified by all teams, but the average team identified only 35.5 of these, and not many teams found the same error. Since the average team detected only 35.5 requirement errors, if only one group were used, as in a normal project, 56.5 errors would remain to be detected during downstream phases, such as coding and testing. Moreover, requirement errors of the greatest severity were identified by the fewest teams. No wonder we have a software crisis.

Problems in software engineering, especially requirement specification, belong to the category of *wicked problems* [Con87]. Wicked problems are those who cannot be solved in traditional systems analysis approach, which is 1) define the problem, 2) collect the data, 3) analyze the data, and 4) construct a solution. The only way to really understand a wicked problem is to have solved it. These problems have no stopping rule. Solutions to wicked problems are not right or wrong, they just have a degree of sufficiency. This is the case with requirements elicitation; one cannot really know what are the client's needs until one has implemented the full requirement specification.

Full discussions of the problems of obtaining good requirements and of the effect of the failure to obtain them may be found in a textbook by Davis [Dav90] and in a paper by Krasner [Kra88].

The importance of obtaining good requirements for software-based systems cannot be understated. Many methods and tools have been devised for analyzing requirements, but only recently has there been a focus on what needs to be done before the analysis can begin, that is, eliciting the information from the client and identifying the abstractions contained in this information.

The following sections give a brief description about requirements engineering in general, emphasizing the sub area of requirements elicitation, which is the concern of this research.

1.2 Requirements Engineering

The traditional software development life cycle consists of: requirements definition, design, coding, and testing (including integration) [Sch92]. Heretofore almost all work in software, both formal and informal has assumed that the requirements were known, and in many cases formally stated. It has become apparent recently that the normal situation is that the requirements are given in incomplete, inconsistent, and vague statements. From these poor statements it is impossible to develop a system; to design, code, and test. Investigation of the software crisis, which was discovered and named in the 1960s [NR69], determined that requirements deficiencies are among the most important contributors to the crisis; in nearly every software project which fails to meet performance and cost goals, requirements inadequacies play a major and expensive role in the failure.

Requirements engineering is the discipline that begins with the customer's statement of need and ends with well-defined specifications to be used for the development. Note that according to IEEE Standard 610.12-1992 for Software Engineering Terminology [IEEE92], the term "specifications" is a written requirement, and will be used in the thesis in requirements documents. The activities involved in requirements engineering are: capture, representation and analysis, and access of the requirements information.

Three disciplines of requirements engineering, which are active through the whole life-cycle of the software, have already been identified [RE93],

1. requirements elicitation for capturing the requirements data,

2. requirements analysis to model and analyze the requirements, and

3. requirements management to access the requirements data in a controlled manner [PCCW93].

Requirements analysis is the most familiar activity in requirements engineering. Most of the existing technologies on requirements have been directed to requirements modeling and how to enhance the semantics of such descriptions (See Section 1.4). These methods and tools assume that the requirements are already written and stated in some restricted language.

The purpose of acquiring or elicitating requirements is to gather enough information to begin to build the conceptual model of the client's enterprise, the model from which it will be possible to write the requirements [WLLO90]. The abstractions identified during elicitation may become the key abstractions of the conceptual model. The expected output of the elicitation phase is a list of well-stated requirements, formal or informal, that can be easily transformed to one of the modeling methods languages.

Requirements management is a new area recognized by the SEI in its Capability Maturity Model (CMM) [PCCW93], in which the concern is the ability to access requirements, to keep changes of requirements under configuration control, and to keep a trace from requirements to all the other work products that implement the requirements, e.g., design documents, test documents, code modules, etc.

Traditionally, a requirements analyst (RA) is needed to sift through these statements and produce consistent, complete requirements, specified enough to be used for the development of the system, e.g. design, code, and testing. As a result of recent research [CWS93] that has identified elicitation as a major step requiring competence, there is a specific subtask of requirements engineering called requirements elicitation, and the person who does it is the requirements elicitor, who may or may not end up being the requirements analyst. Thus, in the rest of this thesis, the requirements elicitor is called simply the elicitor in order to distinguish his or her function from that of the requirements analyst.

This research is focused requirements elicitation. The following section gives a description of what is involved in requirements elicitation.

1.3 Requirements Elicitation

Problems of requirements elicitation are of three kinds,

1. problems of scope, i.e., whether the requirements cover too little or too much,
2. problems of understanding, within a group, such as users and developers, as well as between groups, and
3. problems of volatility, i.e., the rapidity with which requirements change.

Problems of understanding, according to McDermid [McD89], include:

- users having incomplete understanding of their needs,
- users having poor understanding of computer capability and limitations,
- analysts having poor knowledge of the problem domain,
- user and analyst speaking different languages,
- the ease of omitting obvious information,
- conflicting views of different users, and
- the vagueness and untestability of requirements, e.g., “user friendly” and “robust”.

According to Leite, requirements elicitation has received little attention in the past from the

software engineering research community, "..because it is an area where one has to deal with informality, incompleteness, and inconsistency" [Lei87].

Requirements information for systems come in many formats, depending on the customer, problem domain, and extant systems. The information may arise in many forms, such as

- informal technical notes,
- notes from meetings,
- statements of work,
- requests for proposal,
- operational concept documents,
- interviews with customers and users,
- menus of operational features observed in competing or preceding products, and
- functions observed in competing or preceding products, and
- technological surveys.

In the process of gathering requirements for a new system, the elicitor may become inundated with a large volume of data in a wide variety of formats. To promote the likelihood of successful analysis and design of the new system, the elicitor must be able to capture all the information related to the requirements from the client. Thus, the process of capturing the customer requirements involves

a. abstracting the material to identify the main concepts in the customer transcripts,

b. conducting negotiation with the customer in order to verify that the customer need is fully understood, and if not, filling in the missing details, and

c. stating the requirements in simple imperative sentences (This activity is not within the scope of this thesis.).

The purpose of this work is to suggest methods for automatic assistance for the human involved in requirements elicitation in the informal stage. The suggestions include

1. a method to assist in abstraction identification and

2. a method for maintaining the abstractions for negotiation with the customer.

1.4 Existing Requirements Engineering Methods, Tools, and Systems

There has been a modest amount of work in requirements engineering over the past few years ending with that presented at the first International Conference on Requirements Engineering (RE'93) [RE93]. Even among the requirements engineering environments presented at RE'93, none provide any assistance in elicitation and abstraction identification. They all seem to start with abstractions already identified, and focus on organizing the already identified abstractions.

In this section, two types of existing technologies in requirements engineering will be described,

a. methods and tools for requirements analysis, and

b. work in requirements elicitation that shows different ways of attacking the elicitation problem; social perspectives, natural language processing techniques, using a lexicon, and interactive nudging systems.

1.4.1 Tools and Systems for Requirements Analysis

Current tools, methods and systems for dealing with requirement include SADT [RS77, Ros77], IORL [SM85], PSL/PSA [TH77], RDL [EFRV86], RSL [Alf77, Alf79, Alf85], PAISley [Zav82], RML [BGM85], ECSAM [LW89, WLLO90], and Burstin's prototype tool [Bur84]. The first two are graphically oriented, and the second of these is automated.

Structured Analysis and Design Technique (SADT) was developed as a method of modification of all complex systems, data processing or not. A description is made by model diagrams called artigrams. They model processes, inputs, and outputs and enable hierarchical decomposition into sub-processes. SADT uses a graphic language for expressing the requirements of the system under design. An SADT model is an organized sequence of diagrams starting with a top-level overview diagram. A diagram is composed with at most six boxes connected in arbitrary manner with arrows and accompanied with explanatory text. Each lower-level diagram is an explanation of one of the boxes at the parent diagram. An arrow between boxes represents a constraint relation between the boxes and not necessarily a flow of control or data.

Technology for Automated Generation of Systems (TAGS) is a system development method that has the designers focusing on writing requirements rather than on coding. TAGS is composed of the Input/Output Requirements Language (IORL), a tool system, and the TAGS method. IORL is a graphic and tabular language that allows specification of each important software, hardware, embedded, or management of the system under design. The system must be specified as a hierarchical collection of components that interact through data links with a controlling mechanism that dictates how information flows through the system. An IORL specification is a schematic block diagram (SBD). The highest level SBD shows the major system components and the data interfacing between them. Each such component may be expanded in a lower level SBD. Associated with each SBD are a variety of other diagrams supplying other

information about the components in the SBD. These include diagrams for specifying control, logic, and data flow of and among the components of an SBD. The associated tool set allow construction of the diagrams, a number of static checks within and between diagrams, and simulation driven by the diagrams. The multi-view orientation of TAGS is similar to that of SARA, except that the latter is directed at implementation design while the former is directed at specification.

Problem Statement Language/Problem Statement Analyzer (PSL/PSA) is a formal language for the statement of problems that is assisted by the analyzer. PSL is a language for expressing the objects and relations among objects of the system under design. A system consists of objects which may have properties which in turn may have values. The objects may be connected by relationships. The language has a rich collection of standard object, property, and relation type that can be used to describe all aspects of an information system including input, output, data flow, system and data structure, performance, and management. PSA accepts PSL as input, builds a relational data base capturing the content of a PSL specification and prepares a variety of reports. The data base can be used as a living specification that can be interrogated and updated as necessary.

TRW's Software Requirements Engineering Program (SREP) project has produced the Software Requirements Engineering Methodology (SREM) which makes use of a number of tools comprising the Requirements Engineering and Validation System (REVS). SREM was built for dealing with asynchronous real-time systems, but can be applied to general interactive systems. The processes are described by a language called Requirements Statement Language (RSL). REVS is used to translate sentences of RSL into tuples of relational database called the Abstract System Semantic Model. Tools are provided for interrogating and updating this database as well as for preparing reports. RSL differs from PSL in that the former emphasizes flows.

Winchester's Requirement Definition Language (RDL) of UCLA's SARA System also expresses relations between objects of the system under design. Each sentence in RDL represents a tuple of a relational database built up about the system under design. Some of these relations can be expressed pictorially through the use of SARA's other modeling languages for exhibiting system structure, control, and data flows. The multi-view orientation of SARA is similar to that of TAGS, except that SARA is directed at implementation design while TAGS is directed at specification.

PAISLey is a Process-oriented Applicative and Interpretable (executable) Specification Language. The specifications that are generated are operational in the sense that they are an executable model of the proposed system interacting with its environment.

The designers of Requirements Model Language (RML) observe that requirements are meaningful only in the context of certain real-world knowledge. Thus, specification processing tools should have access to a knowledge base that can be consulted to provide implied details that are not given explicitly. For instance, when the requirements specify a temperature and tolerance, knowledge about heat, temperature, measurement, tolerance limits, Fahrenheit, Celsius, etc. is implied on the part of the reader and is often used implicitly by the programmers. An

RML specification organizes the world as a collection of interacting objects. For each object, its characteristics are given as incomplete sentences with the object implied as a missing parameter. Thus, an object's specification can be viewed as a collection of relation tuples with the object appearing at least once in each tuple. A complete specification can thus be organized as a relational database. It is intended to build tools for translating an RML specification into an artificial intelligence language for knowledge representation so that existing tools can be used for extracting the implied information of a specification.

The ECSAM method of requirements analysis and modeling of Embedded Computer Systems and their software has been developed and used since 1980 at the Israeli Aircraft Industry (IAI). The ECSAM modeling approach addresses the conceptual and design models and defines the mapping between them. The conceptual model, also referred to as a *domain model* [KCHNP90], describes the system in the problem space and is a necessary foundation for the understanding of the physical phenomena on which the system is based. Its main purpose is to support the behavioral and performance analyzes of the system which leads to a consistent and precise specification of requirements. The design model describes the system in the solution space. It represents the actual structure of the system and supports the design process. ECSAM also defines the mapping between the conceptual model and the design model. The languages of STATEMATE [HLNPPSST90] are used to produce the graphic models and to represent the ECSAM views. The ECSAM conceptual model is described by the following three views:

1. The logical modules view — describes the partitioning of the system into its logical subsystems, containing the external information that flows between the system and its environment, the information that flows between the internal subsystems, and the functional activities performed by each logical subsystem.

2. The operating modes view — describes the system's main operating modes and the transition between them.

3. The dynamic processes view — describes the behavioral processes that occur in the system in its various operating modes in response to external or internal events.

The overall dynamic behavior of the system is jointly described by the last two views. The three view are complementary and interrelated.

Burstin presents a method for obtaining requirements. He identifies the users of the system and activates them several times in several stages in order to refine and correct the requirements. Computer embedded systems usually operate in real time and are activated by or invoke external processes. These processes may result either from human activity or from a non-human entity like a signal generating source. Therefore, Burstin defines *abstract-users* which are all the entities, human or non-human, that have anything to do with a system, feeding it input or using its output. As Burstin stated, obtaining more users' views will result in a fuller coverage of the system. His goal is to organize the requirements information so that one can scan the information through user views, system views, or process views. However, in this method, all the analysis of the user interviews is done by a human RA (requirement analyst). The problem is that humans

make mistakes and overlook relevant ideas. As shown in one of the experimental results in Section 1, in one particular experiment, each group identified only one third of the errors and none of the severe errors.

Burstin's prototype tool allows tuples of a relation, i.e., sentences, each with a verb and objects, to be organized into a hierarchy of abstractions. Each abstraction contains those sentences sharing a common collection of objects, with the verbs representing procedures of the abstraction. There are tools for introducing and moving sentences to and from abstractions and for placing and moving abstractions in the hierarchy. There is also a rudimentary application-oriented expert system that helps recognize when two or more phrases of sentences may be talking about the same thing, e.g., *plane* and *airplane* or *passenger* and *flier*.

1.4.2 Deficiency of Past Work for Requirements Elicitation

All of these systems are useful for working with sentences and abstractions of a requirements document, once they are recognized and formed. Organizations implementing and using two of these, PSL/PSA and SREM, report much user satisfaction [TH77, Alf85, SSR85].

It is interesting that all the above requirements analysis systems deal with relations and all but the first two, which are picture-oriented, and the designers of PAISLey have gone to the use of relational database for storing the relations. All these system can be used to support an abstraction-based requirement development which leads naturally to an abstraction-based software development [BGN86] or object-oriented software development if one identifies an abstraction as an object.

All these methods and tools assume that the requirements are already written down as if there were nothing to it. Those tools demand from the user highly constrained subsets of English consisting of sentences, each of which states one requirement. Some of these tools, such as PAISley, even help simulate the requirements in the context of their environments assuming all the necessary details are defined. But none of these methods give much help in actually obtaining the sentences in the first place and in recognizing the relevant abstractions, especially in the context of a large client organization. The description of all of the methods either fail to mention how to get the sentences or say something to the effect of "get them and write them down" as if there were nothing to it.

Teichroew and Hershey [TH77] offer that "since most of the data must be obtained through personal contact, interviews will still be required." PSA does help this gathering process in that its "intermediate outputs ... also provide convenient checklists for deciding what additional information is needed and for recording it for input."

Alford [Alf77] says that the "SREM steps address the sequence of activities and usage of RSL and REVS to generate and validate the requirements. It *assumes* [italics are not in the original] that system function and performance have been allocated to the data processor, and have been collected into a Data Processing Subsystem Performance Requirement or DPSPR."

Even eight years later, Scheffer, Stone, and Rzepka [SSR85], from a completely different company which had been using SREM, state only that the "initial input to SREM is a system specification that is translated into RSL and interpreted to determine the interfaces with the outside world, the messages across these interfaces, and the required processing relationships and flows."

The first step of the TAGS method [SM85] is the conceptualization step. "User concepts and requirements are used to develop a conceptual model that is the basis for subsequent engineering." This conceptual model is the top level SBD. In the cited article, there is advice on the issues that should be dealt with in arriving at it. However, no tools are provided as the TAGS method deals with activities that follow the production of this first SBD.

Thus, the gap between the initial fuzzy natural language statements from the individuals in the client organization to the sentences, i.e., relations, with which these tools work is still too large. Methods and tools are needed to close this gap.

1.4.3 Requirements Elicitation Methods and Tools

More recently the software engineering community has been paying attention to the problem of eliciting the raw information from the clients [LPR93]. Some of these works attack the eliciting problem from the point of natural language processing techniques. Those works have attempted to work with problem descriptions in a restricted natural language to produce formal descriptions [SHTUE87, ISK93]. Leite and others have devised methods and tools for building the vocabulary of the problem domain [LF91, LF93]. Ryan [Rya93] discusses the general role of natural language processing in requirements engineering. Others concentrate on developing a good social interaction between the client, the users, and the requirements engineer. Some of this work, e.g., contextual inquiry [HJ90], has focused on observing the client's organization in action and modeling what it does and why. There are works that guide the human user, via user friendly interfaces, in order to obtain the information from him or her. In the following, work of each of the three categories is described.

1.4.3.1 Natural Language Processing

Work was done in Japan [SHTUE87, ISK93] on methods to derive formal specifications from informal ones using knowledge about relations between structures of natural language and structures of the real world. However, the so-called natural language of the informal specification ended up being a very restricted, unnatural language.

Kevin Ryan [Rya93] gives a good discussion about the role of Natural Language Processing (NLP) in requirements engineering. While saying that NLP in the requirements engineering process has been overstated in the past, possibly because of misunderstandings of the requirements process itself, he identifies some phases and tasks in which NLP may be applied.

The task of the requirements elicitor is portrayed as one of translation between the two specialized worlds of the application domain and computing. Mere understanding of the syntax or even the specific semantics of a specialized language is not the most crucial factor in bridging the communication gap. Of far greater significance are the unstated assumptions that reflect the shared, common sense, knowledge of people familiar with the social, business, and technical contexts within which the proposed system will operate. To rely solely on text as a source of knowledge or to expect the client to reduce all his or her demands to a textual form is clearly impractical.

Note that the purpose of language understanding software is to understand sentences without human help. The purpose of abstraction finding tools is to help a human find key concepts; it is intended to be used by humans and with human support. Because an abstraction finder has a human operator, it is quite acceptable if it reports only chunks of words that a human can still recognize. This is not so for a language understander. Also ambiguous sentences pose major difficulties for understanding, but not for an abstraction finder. A language understander tries to optimize on smartness, but an abstraction finder needs to optimize on not losing any abstraction, and reducing the volume to be examining by the human even at the expense of reporting spurious abstractions.

The activity of stating the requirements at the end of the elicitation phase is not new. Burstin defines a requirement as a basic unit of information. This unit is often represented as an imperative sentence with a function as the verb of the sentence and abstract objects as the direct objects of the sentence. Those basic units of information are elicited by interviews with different users of the desired system.

Also, Leite has worked in enhancing the semantics of simple sentences by means of natural language structure called the Language Extended Lexicon (LEL). This LEL contains the vocabulary of the language of the problem. After a process of elicitation, which is not dealt with by Leite, the requirements analysis group should try to achieve consensus over a list of requirements. Based on this list, a lexicon is built. LEL building cannot be automated, and Leite offers only guidelines for the humans doing the task.

1.4.3.2 Social Perspectives

More attention was given lately to the social perspectives of requirements elicitation process. Goguen and Linde survey a variety of methods used in requirements elicitation [GL93], including introspection, interviews, questionnaires, and protocol, conversation, interaction, and discourse analysis. These techniques can elicit tacit knowledge by observing actual interactions in the workplace.

Contextual inquiry [HJ90] is a means of gathering information about a customer's work practices and experiences. The contextual inquiry approach focuses on interviewing users in their own context as they do actual work. The technique contributes to forming a valid understanding of the nature of user work that can be used as a basis for effective design action. The key concepts of the technique are: (1) context of actual experience, (2) partnership in order to

empower the interviewer to articulate, (3) interpretation of the customer's work practice, and (4) focus in order to create a shared understanding.

Contextual inquiry is an adaption of field research techniques taken from psychology and anthropology. Structuring contextual inquiry is a way to synthesize qualitative data into conceptual groupings, using affinity diagrams. Developing an affinity diagram is a social, group process.

1.4.3.3 Guiding Systems

Most application domain tools, that are more concerned with the modeling of the application domain, have a user-friendly interface in order to guide the user in inputting his or her knowledge about the domain [Fea93, Eas93]. Many of these provide a way to organize a domain model into a collection of abstraction nodes into which all information about these abstractions is stored. Finkelstein *et al* [FKN92] use a framework of multiple viewpoints, perspectives, into which to partition the system specification. The framework suggests a restricted structure of partitions and slots, into which the elicitor has to fit in his knowledge about the domain.

There are some tools which are more modelling oriented. READS [Smi93] is an hypertext system designed to support the key requirements engineering activities of requirements discovery, analysis, decomposition, allocation, traceability, and reporting. READS facilitates the construction, browsing, and maintenance of a typed hypertext network with a user interface designed specifically for the system engineer. However, as quoted in the paper, "Requirement discovery and extraction are done by examination of the document through scrolling and regular expression searching. Candidate requirement statements are selected with the mouse and placed into the requirements inspection window from which they may be saved into the project database." Still, the process of capturing the requirements in the first place is done by humans.

RETH, Requirements Engineering Through Hypertext [Kai93], also aims to establish links among natural language requirements statements and the representation of objects in a domain model. It provides a representation mediating between the completely informal ideas of the user in the very beginning and the more formal models and requirements. RETH attempts to support the activities *problem analysis*, and *requirements definition* that can be viewed as one of *eliciting* the requirements from the users. Kaindl, the author of of the paper, suggest "to get help from an analyst (a requirements engineer)" since requirements formation is "too difficult for inexperienced users". RETH only helps to gather and structure the requirements by supporting *brainstorming* through hypertext. It does not automate any part of the elicitation process.

All these methods give the human elicitor technical assistance, but mostly rely on the human to supply all the requirements information according to a strict template.

1.5 Envisioned Requirements Gathering Environment

This work is aimed at producing essential parts of an envisioned integrated environment called REquirements GAthering Environment (REGAE), for gathering, sifting, and writing requirements. This environment may very well be part of a large environment used for software development, deployment, and maintenance [CASE88]. For now, REGAE is described as helping the human requirements elicitor, massage transcripts of interviews with members of a client organization into a consistent, complete, unambiguous, coherent, and concise statement of what the organization wants. No matter what language is being used either for the interview transcripts or for the final requirements, REGAE should support any possibility. Usually the input to REGAE will be a natural language transcript, possibly with pictures [Har87], but REGAE should support any language possibility. The output language in which the requirements are written, can be anything from natural language with pictures, to any of the requirements expressing languages mentioned in Section 1.4.

Since not enough is known about an effective requirements writing in order to be able to codify the process, a completely expert-system approach is excluded, at least for now. Therefore an environment is envisioned, consisting of clerical tools that help with the tedious, error-prone steps of what an elicitor does.

The goal of REGAE is to organize the whole collection of requirements information as a network of nodes, each denoting an abstraction and containing a description of all that is known and required about the abstraction. The arcs between the nodes can be used to describe the relations between the abstractions. Of course, what relations these arcs represent is left up to the elicitor. Certainly, there are arcs that should be generated automatically. For example, should there be a link between all copies of a given sentence and from each word identifying any abstraction in any sentence to the node defining the abstraction? Such a network is intended to be used both in the requirements specification and in the subsequent development.

REGAE needs two basic kinds of tools:

1. to assist identify the abstractions that will make the nodes from the transcripts of the interviews, and

2. to help organize the abstraction into a network of abstraction-nodes, each to contain a consistent, complete, unambiguous, coherent and concise description of that abstraction.

In order to motivate the description of tools of the first kind in the subsequent section, it is useful to understand the envisioned tool of the second kind. This tool provides a medium in which nodes, implemented as windows on a work station screen, can be organized into a network, as suggested by Figure 1 and Figure 2. Each window can be made to hold arbitrary text, including text that causes displaying of a picture. Any arbitrary element of the text of any window can be given links connecting the element to any window or to any element, possibly in another window. Figure 2 shows two windows from a description of an airline reservation system.

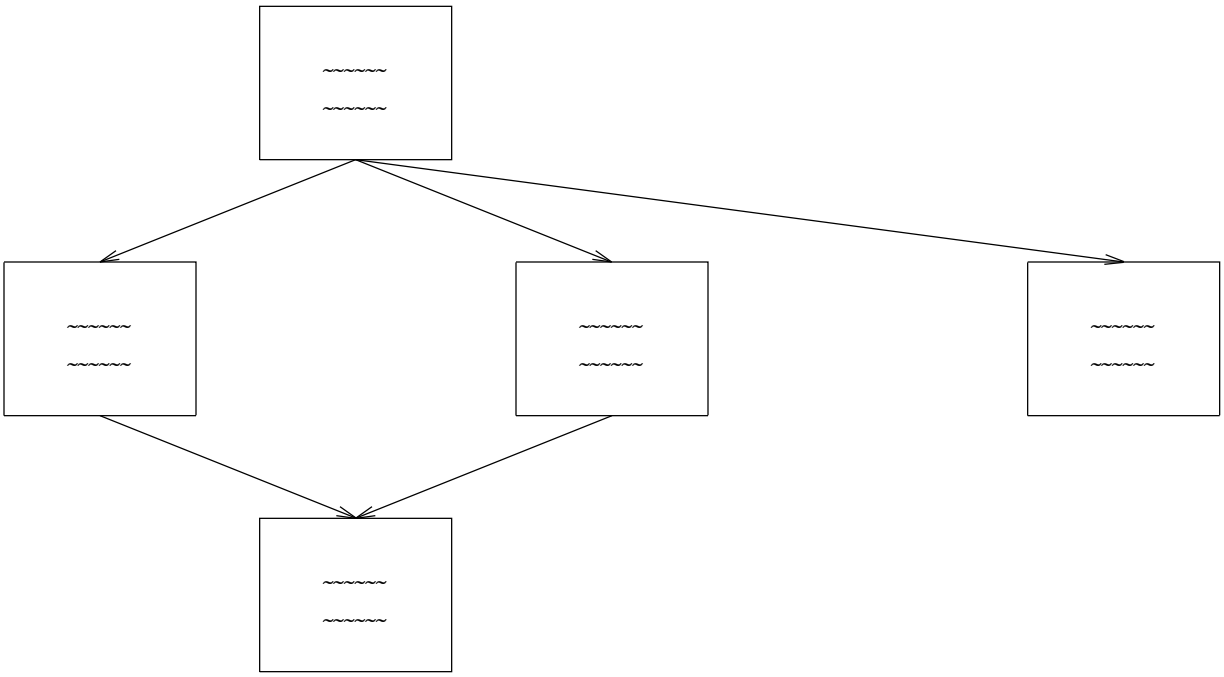


Figure 1: Network of Nodes
ציור 1: רשת של צמתים

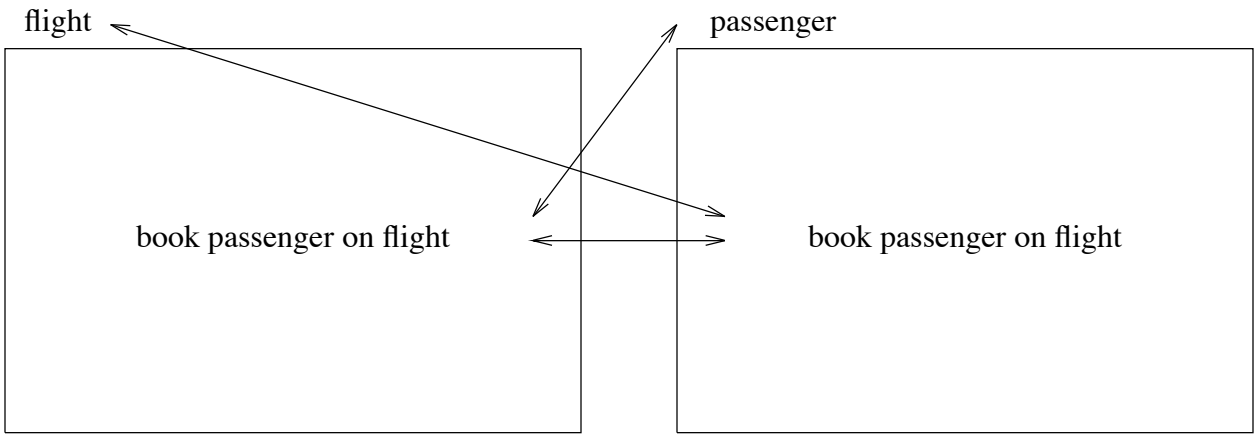


Figure 2: Links
ציור 2: קשרים

The links connect an element to windows giving more details about the element or to other elements talking about the same or related concepts, as the elicitor desires. The elicitor can use these links to navigate through the windows as he or she is tracking down information that allows the contents of each window to be refined into a suitable description of the window's abstraction.

This description of the tool of the second kind suggests building it on top of some existing hypertext system [YMvD85, Con87, DS87]. Indeed, Garg and Scacchi have suggested maintaining all life-cycle documents as hypertext [GS87]. More recently other groups have come to the same idea and have produced functioning hypertext-based requirements management systems [LF93, Smi93, PT93].

1.6 Plan of Thesis

The remainder of the thesis is divided into four sections.

Section 2 presents a key process in requirements engineering, that of abstraction identification. Section 2.1 gives operational definitions and assumptions, and Section 2.2 describes past attempts to build tools for abstraction identification and their weaknesses. Sections 2.3, 2.3.1 and 2.3.2, introduce a new approach for abstraction identification that is the subject of this dissertation. Then Section 2.3.3 describes how the new approach avoids most of the weaknesses of previous approaches. However, as shown in Section 2.3.4, the new approach has its own possible weaknesses, which, it is argued, can be defended against. Section 2.3.5 sketches the program of AbstFinder and Section 2.3.6 analyzes it; finally, Section 3 gives scenarios for using AbstFinder.

Section 4 explains how to evaluate a new method or a tool in requirements engineering, defines criteria for evaluation, and Section 4.1.1 gives an evaluation plan based on those criteria. Section 4.2 describes the case studies conducted according to the evaluation plan. Section 4.2.3.1 compares AbstFinder to findphrases on the industrial-size case study. Section 4.3 evaluates AbstFinder as an indexing tool. Finally, Section 4.2.4 summarizes the results of the AbstFinder evaluation activities and concludes that AbstFinder is an effective tool for abstraction identification.

Section 5 describes a case study in the use of hypertext to build an abstraction network for requirements engineering.

Section 6 draws the conclusions of this research and summarizes its contributions.

2 Abstraction Identification

This section describes past and a current effort to establish automatic assistance for identifying abstractions. It is assumed that the text from which the abstractions are to be identified is available in machine-readable form. First, however, it is necessary to attempt to define *abstraction* so that it will be understand exactly what is supposed to be identified.

2.1 Operational Definition and Assumptions

Abstraction, in general, is ignoring details. When people try to understand a written requirements document, they usually abstract the contents. In this case, abstracting means ignoring enough details to capture the main ideas or concepts in the document. What details are ignored cannot be defined formally, or even informally. However, everyone involved with a project seems to know an abstraction when he or she sees it. Such a definition is not workable. Therefore, operationally, abstraction identification is defined as identifying some words from the written requirements document, and it is hoped that the scheme for selecting the words yields words that help humans to understand the document.

Eventually, when the elicitor feels that he or she understands the text of the transcript by having a list of abstractions, each abstraction identifier will be used for retrieval of the abstraction's contents. An abstraction's contents is all the sentences, collected from different places in the transcript, that deal with the subject of the abstraction identifier, e.g. "communication". The contents of an abstraction as derived from the initial text received from the customer may be ambiguous, incomplete, and inconsistent. Negotiation with the customer will be needed in order to resolve inconsistencies and to add more information in order to obtain useful requirements. Obtention of requirements from abstractions is a laborious activity and lies outside of the thesis.

In any case, an abstraction is not equal to a requirement. According to IEEE Standard 610.12-1990, a requirement is defined as "condition or capability needed by a user to solve a problem or achieve an objective". Thus, an abstraction can be thought of as higher level than requirements. The correspondence between requirements and abstractions is many to many. The importance of the abstractions is that they can serve as an *initial* list for requirements, and be used for the negotiation with the customer.

There are some who say that abstraction identification may be defined as *inverse data retrieval*. Data retrieval is the activity in which one retrieves data according to a known keyword. Abstraction identification is the activity in which one looks for the key concepts to be used for retrieving information about that concept. One of the main concerns in information retrieval [SM83] is the automatic indexing of documents, which consists of producing for each document a set of indices that form a *profile* of the document. A profile is a short-form description of a document, easier to manipulate, and plays the role of a surrogate in the retrieval stage. Recall that in abstraction identification for requirements elicitation, understanding is needed in order to be able to state the raw requirement. In indexing for information retrieval, a profile is a list of keywords that do not necessarily have any meaning and cannot be used as abstractions

identifiers. However, a list of abstractions identifiers is a good list of index terms for retrieval.

Heretofore, abstraction identification has been done manually by an elicitor. The elicitor scans all the transcripts, trying to note important subjects and objects of sentences, i.e., nouns. The problem is that humans get tired, get bored, fall asleep, and overlook relevant ideas. So it is proposed that REGAE contain tools that do the clerical part of the search without getting tired, falling asleep and overlooking anything. The human elicitor still has to do all of the *thinking* with the output of the tools, but he or she will be confident that no piece of information has been overlooked in the process of gathering input to the human process of abstraction identification.

That is, no matter what, the elicitor must read all the input at least once. The larger this input, the more that must be digested in the elicitor's process of abstraction identification. There is the danger of information overload in gathering this input. To avoid information overload, it is useful to somehow reduce the size of the input that must be digested. The danger in reducing the size of the input and relying only on the reduced input is that something important might be overlooked. Therefore, confidence is needed in that the reduced input overlooks nothing important.

The identifiers of the abstractions can also serve as titles of sections of the requirements. Each of these sections has to be filled with details in order to produce a well defined requirement. For instance, the section titled "navigation" might be filled in as follows, "The system shall navigate according to the parameters, how, when, where". Actually, the most refined abstractions are needed for the requirements, in order to give each individual requirement the most accurate title. For example, in the RFP transcript given later, "Unmanned Air Vehicle" is a well defined abstraction and is mentioned in almost every paragraph. However, this phrase is the title of the entire document, which identifies the whole project, and it does not help much in capturing the detailed requirements needed to develop the system. So, a more refined abstraction identifier such as "navigation", "launch recovery", or "communication", which identifies some function or data, is much more useful for a well defined individual requirement.

The list of the abstraction identifiers does not replace the original transcript. Reading only the list of requirement titles will not lead to understanding the client's needs. This list, however, assists the elicitor in two ways. First, it helps the elicitor keep the important concepts in focus. Second, it is used as a reference in order to keep the elicitor from overlooking anything.

Underlying all the approaches attempted in the past and finally taken here are some assumptions that ultimately have to be validated. Their validation will come retroactively as a result of the success of the resulting tools. The assumptions are that

1. at least some manifestation of all abstractions is expressible within the confines of a single sentence and

2. each individual abstraction is discussed in more than one sentence.

If these assumptions hold, then a repetition-based approach, such as proposed below, should work. The main idea behind such an approach is that the importance of a term in the text is

proportional to its frequency of occurrence within the text. It has been empirically verified that a writer repeats important words in the text as he or she tries to explain or verify them [Luh58].

The repetition-based approach implies that important abstractions for requirements are discussed more than once among the sentences obtained from the clients. The rationale behind that is that a requirement has to appear once for definition and at least once for usage. If a requirement is not defined or not used at least once, then it cannot be considered as a requirement to be designed. If by mistake, the client forgot to define the requirement but used it more than once, the elicitor will identify it and will negotiate with the client in order to complete the definition. However, if some requirement was only defined and not used, or not defined and used only once, the elicitor might neglect it too.

The assumptions seem to overlook a high-level abstraction that consists of a concept spread out over several sentences that individually do not expose the concept. Either these do not occur or if they do occur, it is assumed that the human elicitor will notice them as an aggregate of several identified concepts. For this identification to be possible, it must be that each individual subconcept is mentioned more than once so that all of them show up and can be recognized. Our experience has shown that these high level abstractions are not a problem to identify.

One key point that emerged in the consideration of the past work is that it is critical for the tool to have guaranteed coverage, even if it is less intelligent. The lack of intelligence is no real drawback since the human elicitor has to analyze the output of the tool anyway. He or she will provide the missing intelligence. Indeed, there are some advantage to forcing the human to think carefully. However, to be sure that the thinking is supplied with full information, full coverage by the tool is critical. Particularly disastrous is a so-called intelligent tool that makes mistakes and leaves things out in its attempt to be intelligent.

2.2 Existing Abstraction Identification Tools

There is a sequence of increasingly better tools developed to assist in abstraction identification.

2.2.1 Grammatical Parsers

An early idea for abstraction identification, reported in [BYY87] was to use a parser in order to find the nouns. The result was that the few errors it made were distracting and it was more comfortable to find the nouns manually. Ultimately, the idea of using a parser in order to find the nouns for abstraction identification was abandoned, because it did not inspire confidence that it found everything. More importantly, the parser would overlook an important noun because it appears to the parser as a verb. For example, in the phrase “book a flight”, “book” is a verb and not a noun as thought to be by many parsers. Even a better, but still ultimately imperfect, parser does not solve this confidence problem. Finally, the abstractions are often noun *phrases* and not just the words. In the same example phrase, the key concept is “flight booking” and not just “flight”, the only real noun found in the phrase.

2.2.2 Repeated Phrase Finder

A second idea [Agu87, AB90] was to use findphrases, a repeated phrase finder, a repeated phrase finder, a repetition-based approach. Counting isolated words in the text is not sufficient, because a lot of information is lost. In particular, information on the relationships in which words are involved is lost. Therefore, it is necessary to consider the phrases in which the words appear.

In its simplest application, the user provides findphrases with the original text of requirements description and a file containing punctuation and keywords. The punctuation and the keywords are used by findphrases to break the text into sentences. findphrases processes those statements and produces a series of reports. The basic output contains: (1) the input file, written with lines numbered and punctuation keywords overstruck (for bold faced appearance). (2) a frequency-ranked table of repeated phrases, and (3) an alphabetically ordered table of the repeated phrases. Each entry in these tables gives the number of the line in the original text in which the phrase occurs, so each phrase may be examined in its original context to decide which abstraction is really represented by the phrase. A number of options are provided that the user may use to control the parsing of the input text into tokens and phrases, to control the printing of the phrases in the tables of the output, and to indicate which additional tables are to be printed.

There is also a learning process in using findphrases. It appears that there are different characteristic sets of punctuation-keywords and of ignored phrases for each language. Moreover, for each application area there appears to be a characteristic set of ignored phrases. When using findphrases, one should fill the punctuation-keywords and ignored phrases files with common words that are actually important abstractions, but whose presence skews the list and populates it with too much noise for finding the other abstractions.

Aguilera [Agu87, AB90] describes tests of the effectiveness of findphrases in helping the elicitor identify abstractions. The tests involved four examples of program development, each of which had multiple versions of the same program ranging from natural language descriptions, through designs, decompositions, etc., to code.

It was desired to determine if findphrases is effective in helping the elicitor to find, in natural language transcripts of interviews about a system under development, *all* of the abstractions that serve as the basis for requirements, design, and implementation. It was deemed effective if we, as humans, do indeed recognize the same set of abstractions in the outputs of findphrases run (with the appropriate parameter files in each case) on all versions of the same problem. Finding the same set of abstractions in all versions says that the abstractions found in the first version, the natural language description, are sufficient to cover all abstractions that will be needed for all subsequent versions, including the code and that no other abstractions will need to be invented during the design and implementation.

The first experiment is Abbott's example of programming with the help of [Abb83] natural language. This example is the focus of a paper [BYY87] that points the need of this phrase finding tool. For this experiment, three versions of program solution are compared. The first version was written in standard English, the second in an Ada-based program design language, and the third in Ada. The second experiment is the problem of writing the phrase finder itself. In writing the phrase finder, the manual page served as the requirements document. findphrases was run with its own manual page to see if the same abstractions that formed the basis for the modular decomposition used in writing the code are identified from the information provided by findphrases. The third experiment takes Mitchell's text book [Mit84] example of writing sorting program starting from the English statement of the requirements and ending with a Pascal program developed with a structured programming method. Four versions of the program solution are compared, the initial English description, two program design language descriptions, and the final Pascal program. The fourth experiment takes Wiener and Sincovec's text book [WS84] example of writing a spelling checker program. They start with a statement of requirements, develop a modular decomposition for the solution, and produce an Ada program.

While the tests were convincing for the documents considered, none of them considered an industrial sized example in the context of a real-live software development. The question of whether the effectiveness scales up remains.

findphrases was found to be effective in aiding the elicitor to identify abstractions in all stages of life-cycle. However, one particular weakness was noticed. A repeated phrase finder fails to count as a repetition of *book a flight* the phrase *book the flight* since it looks for fixed patterns. Were each of these phrases to appear only once, the concept of *booking a flight* would not show up at all in the list of repeated phrases. In many cases, concepts do not appear as adjacent words but rather a set of words separated not more than a few words. Most of these concepts appear as closely separated pairs of words standing for an agent-object relation. Moreover, this relational information often allows distinguishing between semantically distinct uses of the same word by showing the context from which the word comes.

2.2.3 Lexical Affinities

The third idea [MB88, MB88], is to use *lexical affinities* (LAs) as the atomic unit for identifying major abstractions within a text. An LA stands for the correlation of the common appearance of two items in sentences of the language [Cru86]. For our purposes, the definition was restricted, by observing LAs within a finite document rather than on the whole language. For instance, in this thesis, *requirement* and *analysis* are bound by a lexical affinity. For our purpose, were considered only LAs involving *open-class words* as meaning bearing. Open classes words are nouns, verbs, adjectives, and adverbs whereas *closed-class words* are pronouns, prepositions, conjunctions, and interjections [Hud84].

In order to retrieve the LAs from a document, an extracting tool that retrieves the lexical relations and selects as LAs those appearing the most often was used [MB88]. The LA finder's output is a list of lexical affinities with their associated frequency of appearance within the considered text. For instance, the analysis of the rm manual in the UNIX environment, returns as

the most frequent LAs, the list *(delete file)*, *(file file)*, *(file permission)*, etc. each of which appears three times within the one-page document. Were the manual page taken as a statement of requirements of rm, it was believed that this list of LAs would be of great help for assisting the elicitor in his or her process of extracting requirements. The LA finder could be much improved by accounting for the general context or universe to which the document belongs. This context would allow filtering out such LAs as *(file file)*, cited above.

In order to account for the general context or universe, LAs need to be scaled according to their specific contribution in the given document. As a measure of their contribution, it is proposed to evaluate the *resolving power* of every LA. The resolving power is defined as the power Ω of an LA as a function of its quantity of information and its frequency of appearance within the considered text. The quantity of information represented by a word *w* in a given textual universe is defined as [SM83]

$$INFO(w) = -\log_2 P(w).$$

Thus, if a word "asterisk" occurs once in every 20,000 words, its quantity of information is estimated to be

$$INFO(\text{“asterisk”}) = -\log_2 5 \cdot 10^{-5} = 14.29.$$

In contrast, the word "the" that occurs once in every 15 words, has its information contents estimated to be

$$INFO(\text{“the”}) = 3.9.$$

Drawing from this definition of quantity of information for single words, the resolving power of an LA within a document *d* can be defined as follows.

Let (w_1, w_2, f) be a tuple retrieved while analyzing a document *d*, where (w_1, w_2) is an LA appearing *f* times in *d*. The resolving power of this LA in *d* is defined as

$$\Omega((w_1, w_2, f)) = f \cdot INFO(w_1)\, INFO(w_2)$$

The higher the resolving power of an LA is, the more characteristic it is of the considered document. The best LAs, in terms of resolving power, within a document, represent key concepts of the considered document. These LAs may therefore provide valuable assistance to the elicitor in the process of extracting requirements.

In order to test the effectiveness of LAs in helping the human to find abstractions, the LA finder was tried on the findphrases manual page that was used as a requirements document for building findphrases program. The LA finder found all the abstractions identified by findphrases between the most significant leading LAs.

Again, none of the tests considered an industrial sized example in the context of a real-live software development, and the question of whether the effectiveness scales up remains.

The initial tempt to use an LA finder in assisting the elicitor in requirement extraction looks promising. However, at present the LA finder does not find LAs consisting of more than two words of common grammatical structure, verb-noun, adjective-noun, etc. Of course, findphrases has no problem in finding phrases longer than two words.

2.2.4 Remaining Weaknesses

The early idea of using a parser in order to find the nouns for abstraction identification was not satisfactory, because it did not inspire confidence that it found everything. The parser might overlook an important noun because it appear to the parser as a verb. Even a better, but still ultimately imperfect, parser does not solve this confidence problem.

findphrases and the LA finder have each weaknesses that the other does not have. findphrases finds long phrases but identifies only fixed patterns, whereas the LA finder identifies nonadjacent words but is limited to precisely pairs of words. findphrases cannot identify phrases written in differing orders, for example, "book a flight" and "flight booking" which are not the same phrase but do belong to the same abstraction. The basic LA finder cannot handle phrases of length two whose elements are in different order, but which are grammatical variants of the same root, such as "book a flight" and "flight booking". Neither of them identifies synonyms as belonging to the same abstraction.

The new approach described in the next section is an attempt to get the best of both findphrases and the LA finder.

2.3 New Approach

This section describes a new approach that eliminates many but not all of the weaknesses of the older tools. First a formal statement of the searching is given motivated by a description of what is desired. While the new approach solves most of the weaknesses of the older tools, there are a few remaining.

2.3.1 Motivation

In general, a concept that identifies an abstraction may be a phrase within a sentence. This phrase may be composed of an arbitrary number of words, distributed within the sentence, with arbitrary sized gaps, and may appear in different orders in different sentences. For example, by examining the two sentences, "book ... a ... night flight" and "... flight ... booking", an elicitor will suggest the common concept "book flight" as an abstraction identifier.

It is therefore, desired to determine for any pair of sentences, the set of chunks that they have in common independently of the order of these chunks in the sentences. The chunks in general will be words. However, many times, it is desired that these chunks be words sans suffixes and prefixes in order to capture the commonality in the form of the grammatical root of two occurrences of the same word in different parts of speech. Therefore, it is necessary to allow these chunks to not begin and end at word boundaries. That is, in the two sentences

The flights are booked
He is booking a flight

we wish to find the two chunks “flight” and “book”, neither of which is a full word in both sentences. (The fact that they are in different orders in the two sentences is dealt with below.) The upshot of this desire is that the sentences are considered streams of characters with no particular status accorded to the usual word-ending characters such as blanks and punctuation.

One side effect of ignoring word boundaries is that *noise* can creep into the matching chunks. For example, among

book flight
book funny

the matching chunk is “book f”. Fortunately, the elicitor can ignore the “f” as meaningless. By experimentation, it was determined that attempting to algorithmically excise the noise caused significant material to be lost, e.g., in formulae, variables are significant single-character chunks. Also, we are counting on the intelligence of the human user of the program to recognize meaningful words from the chunks. Sometimes this may be difficult. Among

impossible to see
a possibility seems

the common chunks are “possib” and “see”. The two main problems are illustrated here. Will a human be able to connect “possib” to the correct root “possible”? Will the human be misled to believing that “to see” is a common concept. To assist the human in finding abstractions and avoiding being misled, it will be necessary to print with an abstraction at least a pointer to the sentences involved. Again, algorithmic attempts to avoid these problems, particularly the latter, are fraught with the danger of losing information.

2.3.2 Formal Description

AbstFinder takes the novel approach of considering each sentence as a stream of bytes without any semantics, manipulates those streams, and extracts from them meaningful abstraction identifiers. Thus, the problem of finding common phrases between sentences that identify abstractions, reduces to a problem of finding possibly discontinuous common substreams among the streams. The substreams are the words of the phrase that may be discontinuous in the sentence. But, each substream has to be contiguous, since it might be a word or portion thereof, and a word in a natural language is a run of characters.

Finding common chunks that are in different orders in the sentences may be achieved by comparing one sentence against each of the circular shifts of the other, searching, in each case, for possibly disjoint runs of consecutive matching characters in the two.

Some well-known existing techniques for comparing sequences are to see how different they are [SK83] and can be applied only to sequences of equal length. For example, some methods for comparing sequences a and b are Euclidean distance $\sqrt{\sum_{i=1}^{i=n}(a_i - b_i)^2}$, city block distance $\sum_{i=1}^{i=n}|a_i - b_i|$, and Hamming distance that is simply the number of positions in which the corresponding elements are different. The general approach used in sequence comparison is to seek the appropriate correspondence by optimizing over all possible correspondences that satisfy suitable conditions, such as preserving the order of the elements in the sequence.

Central to the literature on sequence comparison is one basic problem, which is essentially the same in speech processing, macromolecular biology, error-correcting compilers, etc. Given two sequences, the basic comparison problem, roughly speaking, is to find a match, i.e., a trace, alignment, or list of changes, between the elements in the two sequence which require the smallest number of changes, such as deletions, insertions, and substitutions. Most application use the dynamic-programming algorithm, via Levenshtein distance. The Leveshtein distance d is defined as the minimum cost of a sequence of edit operations, i.e., change, delete, insert, that change one string into another. The dynamic-programing method is a recursive technique for finding the distance and the corresponding optimum analysis, i.e., the sequence of operations that caused this optimized matching. In the dynamic-programming method, the two sequences which can be of different length, are put on a matrix, and a match is based upon finding the minimum cost to move through all the positions of the matrix, comparing each character of sequence a to each character of sequence b.

Since abstraction identification involves finding common runs of substring, which constitute a meaningful word or phrase in natural language, it is simpler than the mere general problem of sequence comparison. An abstraction identifier is a common concept which appears, at least partially, identically in both sequences. No edit operations are allowed in a concept that is going to serve as an abstraction title. For instance, “Industry” and “Interest” may be close enough for the basic problem of sequence comparison. But, they are completely unrelated concepts for abstraction identification purposes. Thus, the AbstFinder algorithm is a special case of a dynamic-programming problem that takes into consideration only equalities.

Two parameters are involved in that basic problem of sequence comparison, the distance and the corresponding analysis, i.e., the list of changes that caused the match. In some applications it is the distance, and in others it is the analysis that is of primary interest. In abstraction identification, the analysis itself, i.e., the common subsequence, is of higher interest, because those common subsequences will serve as the abstractions identifiers. Since in abstraction identification, the appropriate correspondence of the same concept in different sequences is not known in advance, the distance is also very important, and is used as a criterion for finding that correspondence.

Let S denote a sentence. Then $length(S)$ is its length, and for $1 \leq i \leq length(S)$, $S[i]$ is the i^{th} character of S. For $1 \leq i < j \leq length(S)$, $S[i..j]$ is the substring of S stretching from $S[i]$ through $S[j]$. Also, if $length(S) \geq 1$, $head(S) = S[1]$ and $tail(S) = S[2..length(S)]$; if, however, $length(S) = 0$, $head(S)$ is undefined and $tail(S)$ is the empty string, Λ. BLANK is the blank character. Finally, $S \,||\, T$ is the concatenation of S followed by T.

If $length(S) \geq 1$, the i^{th} circular shift of S, $CS_i(S)$ is defined recursively.

$$CS_1(S) = tail(S) \,||\, head(S)$$
$$CS_i(S) = tail(CS_{i-1}(S)) \,||\, head(CS_{i-1}(S)), \text{ for } 2 \leq i \leq length(S) .$$

It will often be useful to put a blank at the end of S before circularly shifting S in order that the end of S not form a bogus word with the concatenated beginning of S.

In comparing two sentences it will be necessary to pad the shorter one with blanks to the length of the longer one. Therefore,

$$\text{For } n \geq length(S), pad^n(S) = S \,||\, \text{BLANK}^{n - length(S)} .$$

The special case of padding by one more character will be denoted as simple $pad(S)$,

$$pad(S) = pad^{length(S)+1}(S) = S \,||\, \text{BLANK} .$$

A run in two sentences S and T of the same length is a string of consecutive characters that appears in both sentences in exactly the same position of each such that the character before the run in each differ and the character after the run in each differ. For a run to be significant, it is required that its length be greater than *WordThreshold*, a value that has to be set experimentally as described below. Later, each run obtained from comparing two sentences, one of them a circular shift, will be called a *phrase*, because it can contain several words, which are common to the two sentences.

Suppose that $length(S) = length(T) = n$, $1 \leq i < j \leq n$, and $j - i \geq WordThreshold$. Then,

$$run_{i,j}(S,T) = \bigcup \{a \mid 1 \leq i < j \leq n \textbf{ and}$$
$$j - i \geq WordThreshold \textbf{ and}$$
$$a = S[i..j] \textbf{ and } a = T[i..j] \textbf{ and}$$
$$\textbf{if } i \geq 1 \textbf{ then } S[i-1] \neq T[i-1] \textbf{ fi and}$$
$$\textbf{if } j \leq n \textbf{ then } S[j+1] \neq T[j+1] \textbf{ fi } \} .$$

The right hand side yields a nonempty set only when $S[i..j]$ is a run of significant length in S and T.

Suppose that $length(S) = length(T) = n$. Then,

$$runs(S,T) = \bigsqcup_{i=1}^{n} \bigsqcup_{j=1}^{n} run_{i,j}(S,T) .$$

The abstraction in common between two sentences S and T may be defined to be the set of runs in common in their circular shifts after padding each by one blank to prevent the last word of a sentence concatenated with its first word becoming a spurious word. However, as explained below, for simplicity, the runs are those found by comparing the shorter sentence padded to one more than the length of the longer with the circular shifts of the longer.

Let S and T be two sentences. If they are of unequal length, then let L be the longer of the two and s be the shorter of the two. Otherwise, let L be T and s be S. Let $n = length(L)$. Then,

$$Abst(S,T) = \bigsqcup_{i=1}^{n} runs(pad^{n+1}(s), CS_i(pad(L))) .$$

Consequently, even if S and T are of unequal length, $Abst(S,T) = Abst(T,S)$. Later, the abstraction of a particular sentence S will be taken as the union of all $Abst(S,X)$ for all other sentences X. Thus, there cannot be more abstractions than there are sentences.

From the sentences (not really, but the example has to be kept short!)

file to ignore
the ignored files

the working of the definition causes the sentences to be padded to

```
file to ignoreXXXX
the ignored filesX
```

where "`X`" represents a padding blank, which is really indistinguishable from an ordinary blank. The definition causes the circular shifts of "`the ignored filesX`" to be matched for runs against the padded "`file to ignoreXXXX`", as shown in Figure 3.

The formal description admits of a very straightforward implementation that completely avoids generating and storing of the circular shifts. Basically, the sentence that would be circularly shifted, the longer one, is concatenated to itself after the one blank padding and the shorter sentence is compared for runs with the doubled sentence after positioning its beginning at each successive character of the first half of the doubled sentence. Figure 4 shows the algorithmic rendition of the formal run search shown in Figure 3. Note that it is neither necessary to pad the second occurrence of the longer sentence, nor to pad the shorter sentence.

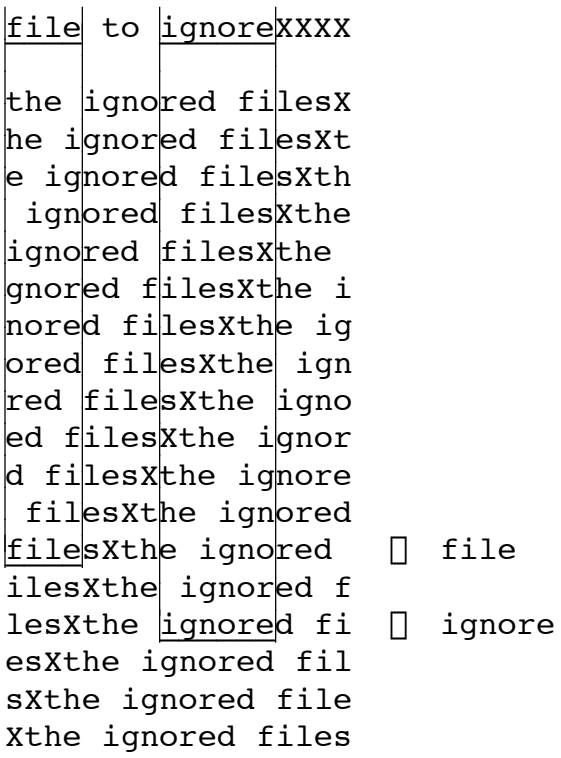


Figure 3: Comparing shorter sentence to circular shifts of the longer
ציור 3: השוואת משפט קצר להזזות ציקליות של משפט ארוך

This algorithm will be recognized as the traditional signal processing algorithm to find commonality in two signal streams [Skl88]. Perhaps the power of this approach comes from its treatment of a sentence as a stream of arbitrary characters with the substreams appearing anywhere rather than being constrained to fall on word boundaries because the sentence is considered a string of words.

It is clear that searching for runs by comparing the shorter sentence padded to the length of the longer with the circular shifts of the longer is different from searching for runs by comparing the longer sentence with the circular shifts of the shorter padded to the length of the longer. However, the difference in the set of runs produced is strictly in what the human would regard as noise (Recall the discussion at the beginning of Section 2.3.). The set of meaningful words and word roots among these runs are the same. The implementation of searching for runs by comparing the longer sentence with the circular shifts of the shorter padded to the length of the longer would require concatenating more than two copies of the shorter sentence in order to simulate its circular shifts; indeed the exact number of copies needed depends on the ratio of the lengths of the two sentences. Given that human intelligence is needed anyway to interpret the runs, and different noise is still noise, for simplicity in the algorithm it was decided to always compare the shorter sentence padded to the length of the longer with the circular shifts of the longer. This

```
the ignored filesXthe ignored filesX

file to ignoreXXXX
 file to ignoreXXXX
  file to ignoreXXXX
   file to ignoreXXXX
    file to ignoreXXXX
     file to ignoreXXXX
      file to ignoreXXXX
       file to ignoreXXXX
        file to ignoreXXXX
         file to ignoreXXXX
          file to ignoreXXXX
           file to ignoreXXXX
            file to ignoreXXXX          □  file
             file to ignoreXXXX
              file to ignoreXXXX        □  ignore
               file to ignoreXXXX
                file to ignoreXXXX
                 file to ignoreXXXX
```

Figure 4: Comparing doubled longer sentence to shifts of the shorter
ציור 4: השוואת משפט משוכפל להזזות של משפט קצר

implies that in a set of sentences only half the total possible comparisons will be done since $Abst(S,T)$ equals, by definition, $Abst(T,S)$.

2.3.3 How the New Approach Avoids Weaknesses of Previous Approaches

The new approach provides an effective way of identifying abstractions in natural language transcripts of client interviews, which allows

1. unlimited phrase length, within the confines of a sentence,

2. phrases with unlimited gaps between the words within a sentence,

3. arbitrary permutations of a phrase to be recognized as the same phrase,

4. automatic matching of subwords that share a common root, when the variation to other parts of speech is regular, e.g., as for “purchased” and “purchase”.

The new approach solves the weaknesses of findphrases and the LA finder algorithms, of being unable to deal with phrases with arbitrary numbers of words, with arbitrary gaps between words of the phrases, and with arbitrary permutations of the words in the phrases. Treating a sentence not as a list of words but as a signal stream frees the algorithm from any phrase constraints. The cyclical sliding of the sentences enables identifying similar words in whatever order they appear in each sentence.

One weakness of all previous methods remains, namely that of identifying as a single concept phrases that have nothing textual in common. There are two manifestations of this, irregularity in changes to other parts of speech, e.g., the past tense of "buy" is "bought", and synonyms. People use different words, called synonyms, for the same thing, and a particular word might appear less used than its concept actually is. Synonyms are used particularly when the requirements are written by more than one person. Both of these problems can be regarded as that of replacing one word by another. Therefore, the program, AbstFinder, containing the basic algorithm, has been provided a facility for synonym replacement, according to a dictionary that can be enhanced by the user.

There is one advantage of findphrases and the LA Finder which is preserved in AbstFinder, their algorithms are independent of the language of the requirements transcripts. Even information that is language dependent, such as ignored words, suffixes, etc., are prepared in a user input file, and is used as is without trying to understand the language.

The tool is purposely non-intelligent so it can guarantee that it considers all of the input and the analyst can have confidence that none of the input has been overlooked as is required.

2.3.4 Possible New Weaknesses of the New Approach

One problem will be to set the *WordThreshold* parameter. If it is not set high enough, then parts of words—called noise in signal processing terminology—might hide the the real abstractions to be identified. With too much noise, the elicitor will not see the trees in the forest and will not find the abstractions. If the *WordThreshold* is set too high, then abstractions that are identified by a word shorter than the *WordThreshold* will be missed. The risk is that to get very meaningful phrases, the threshold may be set too high and not all abstractions will be found. So, it will be necessary to experiment with threshold values, and these values may prove to be different for each problem. It may also be necessary to run the same problem with different thresholds.

A second noise problem can be caused by words or phrases which are meaningful but do not contribute to the abstraction identification process. For each language there appears to be a characteristic set of common words, and for each application area there appears to be a characteristic set of application-dependent keywords.

1. The common words, e.g., "a", "on", "the", "in", etc. obviously do not identify any abstraction. When looking for similarity, these words will skew the list of correlated phrases that identify an abstraction, and will populate it with too much noise for humans

to easily find the real abstractions identifiers. One should fill an *ignored-phrases-file* with common words, in order to mark them for not taking part in the calculation for runs. The *ignored-phrases-file* can also accumulate application-independent words that can be used for any project.

2. The application-dependent keywords are actually important, and repeat a lot in the text. For example, in the RFP text, which is entitled "Unmanned Aerial Vehicle (UAV) and which was used as a case study (See Section 4.2.3), the words "unmanned", "aerial", "vehicle", and "UAV" appear in almost every sentence. When looking for commonality, these words will also skew the list of abstractions making it harder for the elicitor to find other abstractions. So, when using AbstFinder, one should fill an *ignored-application-phrases-file* with these frequent application keywords, which identify larger abstractions.

Filling these ignored phrases files requires experimentation and is basically a learning process. This process is described in Section 3.

A third noise problem can be caused by common long suffixes. For instance, "cation" in "application" and in "communication", or "ance" in "accordance" and in "appearance". In order to avoid these suffixes being counted as possible abstractions by AbstFinder, one should fill the *ignored-suffixes-file* with the recognized common suffixes, in order to mark them for not taking part in the calculation for runs.

An enhanced operator $Abst(S,T)$, $Abst_{\Box}(S,T)$ is applied in AbstFinder program, which uses $run_{\Box,j}(S,T)$, instead of $run_{i,j}(S,T)$, where

$$run_{\Box,j}(S,T) = \bigcup \{a \mid 1 \Box i < j \Box n \textbf{ and }$$
$$j \Box i \Box WordThreshold \textbf{ and}$$
$$a = S[i..j] \textbf{ and } a = T[i..j] \textbf{ and}$$
$$a[i..j] \text{ is not an ignored suffix } \textbf{and}$$
$$\textbf{if } i \Box 1 \textbf{ then } S[i \Box 1] \Box T[i \Box 1] \textbf{ fi and}$$
$$\textbf{if } j \Box n \textbf{ then } S[j+1] \Box T[j+1] \textbf{ fi } \} .$$

This means that a run cannot begin with an ignored suffix.

In general, a little noise that sometimes causes useless information to appear among the valid abstractions, does not harm abstraction identification. If there is not too much noise, the elicitor can easily distinguish the the noisy strings from the meaningful words and ignore them. Also an ambiguous phrase, without a common word that was ignored, poses no problem for a human to interpret if it is reported as a repeated phrase.

There is also the problem of inconsistency, of client using misleading concepts. Either using the same concept for different purposes, abstractions, or using different concepts for the same abstraction. For instance, the author has experience as a requirements elicitor of a communication system in which the concept "frequency" was used for frequency of hopping for

anti-jamming purpose, and for the frequency of a clock, which are completely different abstractions. Those inconsistencies originating in the client's transcript are lit up by the new method, which helps identify them and gets the client involved in solving them.

2.3.5 AbstFinder Program

The AbstFinder program incorporates the algorithm described in the previous section. AbstFinder's algorithm uses the information yielded by *Abst*(*S*,*T*) for all distinct combinations of two sentences *S* and *T*. A sentence is not compared with itself, but no attempt is made to avoid comparing a sentence to another sentence that happens to be a duplicate. Indeed, such duplicates should strengthen the frequency of the abstractions embodied in the sentences, especially if they come from different sources. The set of runs returned by an invocation of *Abst* on one pair of sentences, one being a circular shift, is called the *phrases* of that pair of sentences. Whatever meaning can be ascribed to this phrase is the *abstraction* embodied by the phrases in common in the two sentences. The main data structures of the program are corr_phrases and corr_lines. Each entry in corr_phrases is the set of phrases obtained by comparing one sentence to all circular shifts of all of the other sentences; any sentence for which no phrases are found does not have an entry in corr_phrases. corr_lines is an array indexed the same as corr_phrases, such that for each entry in corr_phrases there is an entry in corr_lines that contains the line numbers of the sentences that compose the abstraction that is identified by the corr_phrases entry.

```
Accept four input files:
   a punctuation-keyword-file, an ignored-phrases-file,
   an ignored-suffixes-file, an ignored-application-phrases-file,
   and a synonyms-file;

Partition the text into sentences one per line, where a sentence is
   the text lying between two consecutive elements of the
   punctuation-keyword-file;

comment line and sentence are used interchangeably from now on tnemmoc

Remove from the text strings found in the ignored-phrases-file
   and strings found in the ignored-application-phrases-file, and
   mark suffixes according to the ignored-suffixes-file, and
   replace words by their synonyms according to the synonym-file;

declare N := number of lines; comment = number of sentences tnemmoc
declare corr_phrases[1:N], corr_lines[1:N];
declare NA := 0; comment number of abstractions accumulated so far
   which must always be less than or equal to N tnemmoc
for i from 1 to N do
   corr_phrases[NA] := □;
   corr_lines[NA] := {i};
```

```
    for j from i+1 to N do
      if Abst(line[i],line[j]) ≠ ∅ then
        corr_phrases[NA] := corr_phrases[NA] ∪ Abst(line[i],line[j]);
        corr_lines[NA] := corr_lines[NA] ∪ {i} ∪ {j}
      fi
    od
    if corr_phrases[NA] ≠ ∅ then NA := NA + 1 fi;
od;
NA := NA - 1; comment correct overshoot tnemmoc

comment sort the NA identified abstractions so that the most
    refined ones are at the top of the list tnemmoc

Sort both corr_phrases and corr_lines so that correspondence between
    corr_phrases[i] and corr_lines[i] is preserved and
    the elements of corr_phrases are ordered mainly by increasing
    numbers of phrases in the elements and
    within the group for any number of phrases, by decreasing
    numbers of lines/sentences from which the phrases came;

Prepare and print the output as described below;
```

The output of AbstFinder comes in two parts. The first part is a table summarizing the identified abstractions, and the second part gives a full description of each of the abstractions. Figure 5 shows the first part of an AbstFinder output that features in a later discussion.

```
|------|-------|-------|-------|---------------------------|
| #)   | abst# | corr_ | corr_ |  correlated-phrases       |
|      |       |phras# |lines# |                           |
|------|-------|-------|-------|---------------------------|
| 1    | 42    | 1     | 11    |  punctuation keyword file| |
| 2    | 14    | 1     | 2     |  whitespace|              |
|      |       |       |       |                           |
```

Figure 5: First part of AbstFinder output

ציור 5: החלק הראשון של הפלט של AbstFinder

There is one row in the table per identified abstraction. The first field, labeled "#)", gives a serial number for the abstraction. The field labeled "Abst#" gives the abstraction number assigned by AbstFinder to its first phrase (the NA of the algorithm). The "corr_phras#" field gives the number of distinct phrases that were united into the abstraction by AbstFinder. The "corr_lines#" field gives the number of distinct lines or sentences that contain these phrases.

Finally, the "correlated-phrases" field shows the phrases themselves with vertical bars in between them and after the last one. Each blank starting from the second column after the beginning of the field is significant and is part of its run. This field is truncated by its flowing beyond the physical width of the paper. This truncation is a design feature, and it's purpose is to signal to the elicitor reading the summary part, that this abstraction is identified by too many phrases, and may be too broad (See Section 3). Even if the phrases are truncated, the full list may be found in the corresponding entry in part 2. The abstractions in the table are listed in order of increasing numbers of the correlated phrases, and within any particular number of correlated phrases, in order of decreasing numbers of sentences from which the phrases came. Note that the elicitor uses only the "correlated-phrases" field in order to decide on abstraction identifiers. All the other parameters are for research purposes (See Appendix A).

An abstraction identified by one phrase is more distilled than one that is identified by more phrases. Obviously, there exists an abstraction that identifies the whole document and contains every sentence in the document, but we are not interested in it. So, the first criterion for ordering the abstractions is in order of increasing numbers of correlated phrases. Then, when two abstractions have the same number of correlated phrases, the second criterion for ordering is in order of decreasing numbers of sentences from which the phrases come. The more sentences contained in an abstraction the more significant it probably is.

The second part of the output of AbstFinder is a full description of the abstractions in order of their serial numbers in part 1. Figure 6 shows one of them. The output itself is in the Courier font and the commentary is in the Times Roman font.

```
{1} abst_id=42
====================
correlated phrases of abstraction are
(#=1)                        number of correlated phrases
     punctuation keyword file|      phrases themselves

correlated sentences of abstraction are
(#=11)                       number of correlated sentences
     44 2 8 14 20 21 23 24 30 35 49      identity numbers of sentences
```

Figure 6: Second part of AbstFinder output
ציור 6: החלק השני של הפלט של AbstFinder

Another byproduct of AbstFinder is the *corr-phrases-file*. The *corr-phrases-file* contains a list of all and only the abstraction identifiers, i.e., the correlated-phrases of part 1 of the output.

The full program can be thought of as a kind of clustering [Sal89]. In clustering, one starts with each object in a separate class. Then a distance measure is selected. The next step is to group into one class all objects whose distance is according to a predefined criterion. This repeats until intra-class distances are low and inter-class distance is high.

In AbstFinder, as with normal clustering techniques, there is a similarity measure (the length of the runs among two sentences) and a criterion for deciding when two items are similar (the sum of lengths of the runs being greater or equal to *WordThreshold*). However, true clustering puts each object in one and only one class, as it is a partitioning. In AbstFinder a sentence is allowed to be in several abstractions. Moreover, true clustering starts with an arbitrary classification, and then moves objects from class to class until the criterion is fulfilled. The final result of classification can be heavily influenced by that arbitrary first classification. Generally, in requirements elicitation, no *a priori* classification is available. Moreover, it is desirable to avoid being influenced by any initial prejudices.

Moreover, AbstFinder can be compared only to flat clustering. Hierarchical clustering is not relevant to abstraction identification. The hierarchical clustering may be good for search in libraries [Maa89] where one start at a certain top level and wants to get to the lowest level with clusters consisting of single elements in order to fetch a certain software module. In abstraction identification, one is looking for some middle level in which an abstraction is defined by a good phrase, consisting of a few words in a few sentences, but not too high because an abstraction has to be specific. By using AbstFinder, hierarchies are generated only when the elicitor wishes to do so. The elicitor expresses this wish by zooming into an abstraction that he or she thinks that is too broad (See Section 3).

2.3.6 Performance Analysis of Program

This section analyzes the performance of AbstFinder from three different points of view,

1. an implementation independent analysis of the value of *WordThreshold*,
2. a complexity analysis of the current, prototypical implementation, and
3. a complexity analysis of possible alternative, production implementations.

2.3.6.1 Value of *WordThreshold*

As mentioned in Section 2.3.4, the *WordThreshold* parameter must be set very carefully. An abstraction is identified by a concept, and a concept is composed of natural language words. Thus, if we assume that the minimum length of a meaningful word is three characters, then *WordThreshold* has to be set to at least three in order to be able to capture common concepts as abstraction identifiers according to AbstFinder.

However, while prototyping the tool, It was decided to keep all input spaces between words, and to take them into consideration while calculating similarity. So, a threshold of three characters was found to be too low. It happened often that a string of form "x y" was found as a match. This match is meaningless because "x" is the last character of one word and "y" is the first character of the successive word. So, the threshold was raised to 5 characters, and AbstFinder appeared to capture only meaningful phrases. Of course, each application can have its own *WordThreshold*, and it will be necessary to experiment with the value of the threshold. Fortunately, there is no reason that several different activations of AbstFinder, each with a different *WordThreshold*, cannot be used by the elicitor for abstraction identification.

2.3.6.2 Complexity of Current Implementation

The current implementation is a prototype. It was intended to be, and in fact was, modified often as the exploration of its use showed the necessity to do so. Consequently, the current implementation was designed to be as simple as possible to make it as easy as possible to change. This simplicity ended up costing a lot in run time, but in fact was not too bad in terms of its space consumption. The time and space complexities are considered in turn.

2.3.6.2.1 Time Complexity of Prototype

Given that N is the number of sentences in the input document and n is the maximum length of a sentence, the time complexity of AbstFinder is $O(n^2 \cdot N^2)$, or $O(K^2)$, where K is an upper bound on the input size. Each of the N sentences is compared to the other $N-1$ sentences. Each comparison of a pair of sentences involves comparing the at most n characters of one to the at most n characters of each of the at most n circular shifts of the other.

In principle, sentence lengths are unbounded. However, since they are natural language sentences, they can be regarded as bounded, say, at about 250 characters. Moreover, as the elicitor follows the iterative process of using AbstFinder, the sentences are getting shorter as the ignored files are getting bigger (See Section 4.2.3). Therefore, the time complexity of AbstFinder can be regarded as $O(N^2)$.

The use of AbstFinder with real-life industrial transcripts (See Section 4.2) showed that the time performance is more than acceptable to the elicitor. This acceptability was fortunate because the real performance problem turned out to be that of space, and more than once, opportunities to save time were sacrificed to get a program that fit in the available memory.

2.3.6.2.2 Space Complexity of Prototype

As mentioned, the bottle-neck of AbstFinder is space. The following calculations show how much memory AbstFinder consumed for the RFP case study (See Section 4.2.3).

In general, AbstFinder generates information into two main data structures, corr_phrases and corr_lines. corr_phrases accumulates for each identified abstraction, all the common concepts that identify it, and corr_lines accumulates for each abstraction the line numbers of the sentences that compose the abstraction. If the input document contains N sentences, then in the worst case, N abstractions are possible, i.e., each sentence contains a new concept. For each abstraction, the maximum length of the string that identifies an abstraction is n, i.e., the size of the longest sentence. For each abstraction, the maximum number of sentences that compose it is also N, i.e., if some sentences were to talk about all concepts or there were only one concept. So, corr_phrases's maximum size is $O(N \cdot n)$, and corr_lines's maximum size is $O(N^2)$. Thus, if we regard n as constant, the space complexity of AbstFinder is $O(N^2)$.

In the RFP case study, the input transcripts contained about 2000 sentences. Thus, for $N = 2000$ and $n = 250$, about 5MB memory space was needed to run AbstFinder. Since the prototype was developed and run in a multi-user system, memory was not allocated in advance, because then the program would have been put at the lowest priority. So, memory was allocated dynamically for the space needed for the corr_phrases string generated for each abstraction. Working with dynamic allocation caused problems occasionally when dynamic memory consumers interfered with each another.

All the above is the space needed for AbstFinder output only. This does not include the original sentences of the input transcript. Because of system constraints, the original sentences were kept in a file and not in memory. So, for each iteration of calculating one abstraction, the sentence i that was compared to all the others was brought into the memory, and for each comparison to some other sentence, that sentence was read from the input file. So, only two sentences of original input file were kept in memory at any time. Only by keeping all of the original sentences in memory, and eliminating input of the sentences from a file for each comparison would the time performance improve.

2.3.6.3 Alternative Implementations

There are faster algorithms based on the use of tries. The time complexity of these algorithms can be made $O(N)$ if desired. A trie is essentially an M-array tree, where M is the size of the alphabet used in the text to be searched. The nodes of the tree are M-place vectors with components corresponding to characters of the alphabet. The name "trie" comes from its being used in information re*trie*val. This data structure is very useful for improving searching methods. For a complete description of tries, please see Knuth's book on searching and sorting [Knu73].

The basic AbstFinder step is comparing all the circular shifts of one sentence to all circular shifts of all the others. Thus, in order to save the time of circularly shifting all the sentences, one might consider building a trie encoding all the circular shifts of all sentences. There is one path root for each of the M characters. A path is rooted at its first character and follows one circular shift. A branch in a path represents where two circular shifts that share their initial parts begin to differ.

Each path is of maximum length n (sentence length), and there is at most an M way branch at each node. Also each node has enough additional information that tells from which sentences the various paths that come through it come. Actually the sentence that is compared to the others is followed sequentially, without taking advantage of its representation in the trie. Because all sentences will be in the trie, care has to be taken not to match a sentence to itself.

Suppose one builds a trie of all the circular shifts of all the sentences of the document. It's clear that if one can build a trie capturing all sentences one can build a trie capturing all circular shifts of all sentences. After all, each circular shift of a sentence is a sentence. Therefore, to build the circular shifts of all sentences, one could simply make all the circular shifts first and then build a trie of a much larger set of sentences. Obviously, it is more time effective to add all the circular shifts of one sentence as that sentence is being added to the trie. If the sentence is of length n, each letter, is added to n paths all at once.

Once the trie of all circular shifts of all sentences is ready, the search for matches is linear in the number N of sentences. While sequentially visiting the characters of each sentence, *all* circular shifts of *all* sentences are checked simultaneously for matches.

Note that a substring that is common among some sentences is the shared beginning of more than one path from its first letter. These shared substrings are the runs that identify abstractions.

2.3.6.3.1 Complexity of Alternative Implementations

Thus, once the trie is built, the time complexity of AbstFinder is $O(N \cdot n^2)$. For each of the N sentences, the sentence of maximum length n will be scanned as a trie path of the same maximum length is followed. The time complexity for building the trie is also $O(N \cdot n^2)$. Therefore the time complexity of the alternative AbstFinder is $O(N \cdot n^2)$. If we accept the earlier argument about n effectively being constant, then the time complexity of the alternative AbstFinder is $O(N)$.

The trie has paths of maximum length n rooted at the M characters, and at each step along the way each node has a potential branch of M. Each node also contains information announcing if it is the end of a circular shift, maybe the circular shift itself, and pointers to the sentences involved. Therefore, the space needed for a trie encoding all circular shifts of all sentences is bounded by $(c \cdot M^n)$. It is clear that the trie implementation of AbstFinder requires significantly more space that the prototype implementation of AbstFinder.

It is clear that the upper bound will be never be reached, since we are dealing with natural language in which the number of combinations of characters is limited. Thus, in the following, a more realistic analysis is given.

In the RFP case study, the number of sentences is $N = 2000$, and the maximum length of a sentence is $n = 250$. So, the number of circular shifts of all sentences is $N \cdot n$ where each circular shift is of length n. This results that the number of characters of all circular shifts is $N \cdot (n^2)$, which is the number of nodes needed in a trie representing all the circular shifts. For each node, a trace is to be kept to the sentence in which that specific path is contained. The space needed for trace in each node is 400 (maximum number of sentences in an abstraction) in the RFP case study. Thus, the space needed for an optimal trie representing the RFP document is around $2000 \cdot (250^2) \cdot 400$ or about 50 GB. Even this amount of space is not realistic.

The space that AbstFinder needed for generating the results is relevant for trie architecture too, since it is used for output. In the implementation used for the thesis, the original file was kept as is and only two of its sentences were brought into the memory on each iteration.

2.3.6.3.2 Other Alternatives

Based on the observation that runs show up in the form of shared initial subpaths in the trie, some other algorithms are suggested. Specifically, it is possible to recognize the runs during the construction of the trie and eliminate the matching stage. However, since both stages are $O(N)$, the savings is only in the multiplicative constants.

A Patricia tree [Knu73] represents an optimization of tries. The basic idea of a Patricia is to build a binary trie, but to avoid one-way branching by including in each node the number of bits (characters) to skip over before making the next test. However, this change does not change the order of magnitude of space needed.

2.3.6.4 Final Complexity Assessment

Experience with AbstFinder on an industrial-sized example shows that its real performance problem is space; and this problem is only exacerbated when the faster algorithms are used. In any case, the maximum run time for the industrial-sized case study was three hours. This time was deemed tolerable because it is no problem for the elicitor to go out to lunch or do something else while waiting for it to report on a large input. Moreover, generally speaking AbstFinder is run only on early documents only for the purpose of assisting in identifying abstractions. Therefore, the slow runtime of AbstFinder on large files is no real burden.

Since the first version of AbstFinder was implemented as a prototype, the main concern was simplicity due to expected changes during the research. Indeed the prototype was changed three times. Analysis of the prototype implementation shows that its time complexity is only $O(N^2)$ still reasonable, while space complexity is reasonable too. The trie based implementations are considerable faster, but require unrealistic amounts of space. The conclusion is that the simplest implementation is reasonable both in time and space.

3 Scenarios for Usage of AbstFinder

With any scheme of automated assistance, scenarios for usage should be defined. Typically, in the process of abstraction identification, even the most intelligent elicitor cannot abstract a full transcript all at once. Usually, the elicitor reads some pages in order to learn the terminology of the transcript. Then he or she reads the transcript several times, in an iterative learning process, capturing another set of abstractions in each pass. Thus, an automated assistance for abstraction identification has to be compatible with human capabilities of learning and understanding new material.

This section describes typical scenarios that an elicitor might follow in order to have AbstFinder help identify the abstractions in a new problem given to him or her by a client. It is assumed that the elicitor has on line what the client believes is a complete description of the system to be built. This description is written mostly in some natural language.

3.1 Learning What Words to Ignore

First some trial runs need to be done on small parts of the transcript, taken from different sections of it, in order to learn the language of the document. Learning here consists in identifying the ignored words, putting the common words into the *ignored-phrases-file*, the special application words into the *ignored-application-phrases-file*, and the suffixes into the *ignored-suffixes-file*. The *ignored-phrases-file* and the *ignored-suffixes-file* are accumulated from one application to another. The *ignored-application-phrases-file* is specific to an application. Actually, the *ignored-application-phrases-file* may contain very important high level abstractions that have to be taken into consideration by the elicitor, but which have been recognized, noted, and put into that in order not to clutter up the output. After each run of AbstFinder, the ignored words and suffixes files are updated, because after any change, new noise appears. This process converges after a few runs.

3.2 What to Do with a Well-Organized Document

If the transcript to be analyzed is a well-organized document, using it as is with AbstFinder it may cause some of the abstractions to be concepts that belong to the table of contents or the meta-language that defines document writing. These included organizational concepts such as “summary”, “confidential”, and “base-line configuration”.

The table of contents itself, although it looks like a good classification of the material, is only an organizational list. It is not necessary that each title in the table imply an abstraction. For example, the title “Characteristics” does not identify an abstraction because it is too broad and unfocused. We aim to have abstractions of only functional or informational strength, the two highest module strengths, according to Myers [Mye79].

So, the titles in the table of contents do not necessarily have to appear in the AbstFinder result list. In fact, it is suggested to remove the table of contents from the transcript before applying AbstFinder. Unless the table of contents is removed, every title of it will appear in the abstraction list, because each title appears at least twice; once in the table and again in the specific section.

On the other hand, for an unorganized collection of documents, the list of abstractions produced make good candidates for sections of an organized document produced from their contents and the phrases of these abstractions might very well end up being the section titles that show up in the table of contents, along with the organizational titles such as "Introduction", etc.

3.3 Zooming

Sometimes, important abstractions that are identified by one word, such as "testing", are too general and not very helpful. Moreover, later in the AbstFinder output list, more specific sub-abstractions such as "built-in test" and "acceptance test" may be found, which are more to the elicitor's liking (See Figure 7).

In order to obtain full coverage of all the sub-abstractions concerned, the elicitor can *zoom* into each abstraction that is found to be too broad. Zooming is a refinement of an abstraction by its sub-abstractions. Zooming is very useful for abstracting requirements, and in gaining confidence in the tool.

The following describes the zooming procedure:

1. The first step of zooming projects into a file *f* all the sentences from the original document that contain approximately the candidate string e.g., "test". The file *f* should contain all that was ever mentioned about the candidate string in the original document. The reason for using the approximate grep program, agrep [WM91], is to allow other parts of speech for the word to be found. This requirement is not critical in this particular case because "test" is the root and its changes to other parts of speech are regular. In the case of a root with irregular changes, the elicitor will have to build a more complicated search pattern or search several times to capture all the variations of the root.

2. The second step activates AbstFinder on the file *f*. The result of this abstraction identification is a detailed list of sub-abstractions of the abstraction identified by the candidate string. For the example, this zooming process would provide a better resolution of all the concepts that relate to "test", namely "test equipment", "acceptance tests", "rejection/retest", etc.

Thus, zooming is very useful to the elicitor for focusing on specific abstractions, and to extract their internal sub-abstraction structure fully.

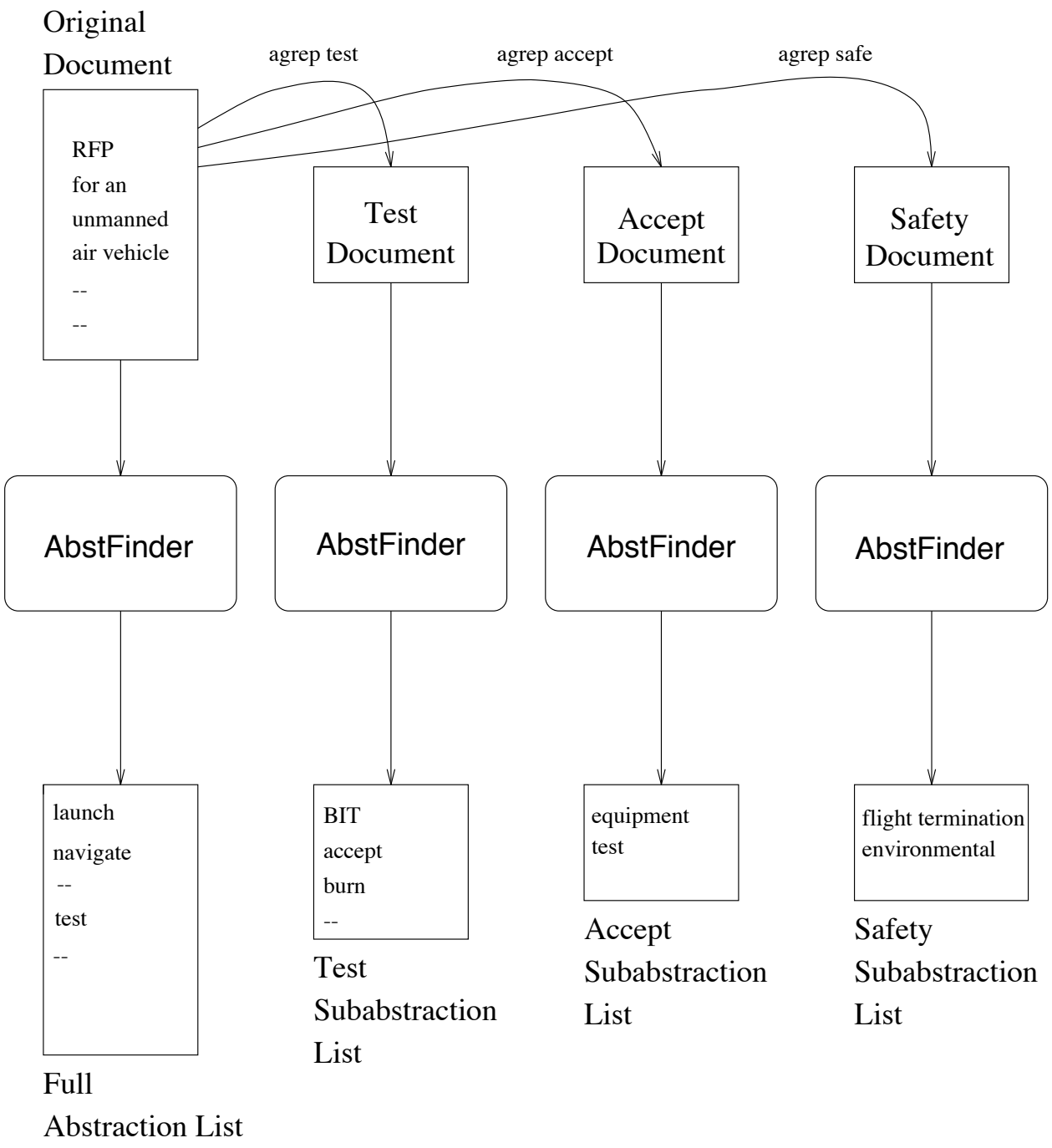


Figure 7: Finding sub-abstractions by zooming
ציור 7: מציאת תת-מופשטים באמצעות הגדלה

AbstFinder identifies the abstractions with no hierarchies. The formatted output of AbstFinder is arranged so that more refined abstractions are higher in the output list. The fewer phrases identifying the abstraction the more refined it is. In reality, all abstractions are equally important. An abstraction that appears at the end of the list may not be any less important. On the contrary, this abstraction was defined among many other concepts and it is difficult to distinguish it from the others. Also the abstraction does not appear many times in the transcript. Thus, it is very important to identify that abstraction as quickly as possible in order to get back to the client and obtain more information about it.

3.4 Iteration to Final List of Abstractions

When using AbstFinder with huge transcripts, the elicitor should read the output list of abstraction and note the abstractions identified by fewer than four or five phrases. Abstractions identified by more than five phrase are difficult to understand. They are also often extraneous because they capture concepts that are too general to be useful. The extreme example is the one abstraction that identifies the whole transcript, and that abstraction is clearly not very useful. Therefore, the elicitor has to stop at some point, a point before which the abstractions are still useful, and at and beyond which they are not useful. It may be impossible to find a single point meeting both criteria, so often the elicitor has to settle for a point at and beyond which they are not useful.

The main purpose of the clerical tool is to identify *all* meaningful abstractions. Without full coverage, the elicitor will never trust the tool to not overlook something important. So if the list is cut off just before the point at which abstractions are identified by six phrases, the concern is whether there are any abstractions that are not recognized because they are identified by more than five phrases. In order to eliminate any worry about a possible lost of abstractions, the iterative procedure described below should be carried out.

This procedure uses the Strainer program, an auxiliary tool (See Appendix B) designed to strain out words from a text file according to a list of words contained in another file. The extraneous abstraction elimination is done by activating Strainer using the logged *corr-phrases-file* as the input list of words to be removed from the original document. Strainer removes a chunk from the text file if and only if it has white space on both sides. Being surrounded by white space means that the chunk is a whole word in the text file and not part of a word. Note that AbstFinder is designed to find common runs of sentences, even if they are only parts of words. Strainer however, is designed to remove only whole words. This feature is very important for the iterative usage of AbstFinder, in order not to ruin the text and keep what remains after straining potentially meaningful.

The following describes the iterative procedure:

1. Activate AbstFinder once on the original document.

2. With the Strainer program, remove from the original document the abstractions already recognized and logged by the elicitor, leaving what is left in another file *f*.

3. Activate AbstFinder on *f*. The result of AbstFinder is a new list of abstractions, without the ones that were recognized before, but with some that had been buried in the first abstraction list after the cut-off point.

The process repeats until finally the elicitor is left in *f* with a list of abstractions, which are all meaningless (See Section 4.2.3). That meaningless list indicates that all the meaningful abstractions were identified previously and strained out from the transcript.

This iterative way of applying AbstFinder and then Strainer, is suitable for a human elicitor to capture large amounts of information. Doing it step by step allows him or her to look each time over a limited, readable, and understandable amount of information and to accumulate it. The elicitor is confident that nothing is overlooked, because things that have not been seen yet will pop up in some later iteration. The iterations continue until finally she or he is sure that the document has been wrung dry of abstractions.

Recall that AbstFinder generates also the *corr-phrases-file* file that contains the abstraction identifiers, each abstraction per line. The elicitor, by using part 1 of the formatted output of AbstFinder, decides which abstractions are meaningful, and stops at abstraction *n*. The elicitor keeps only the *n* first lines of *corr-phrases-file* file, i.e., the identifiers of the *n* recognized abstractions, and removes the rest of *corr-phrases-file*. These *n* abstraction identifiers are also logged in the accumulative *abstraction-phrases-file*.

This accumulated list of meaningful abstractions provides full coverage. The iterative process is illustrated in Figure 8. In this figure, each box labeled “Corr-Phrases” represents one output from AbstFinder and has an iteration number, the number of abstractions accepted by the elicitor as meaningful above the dotted line cut-off point, and the total number of abstractions in the file, reported by AbstFinder, below the dotted line. In the case of the box numbered “#4”, no abstraction was accepted as meaningful, so there is no dotted line, and the number of abstractions there is the total reported.

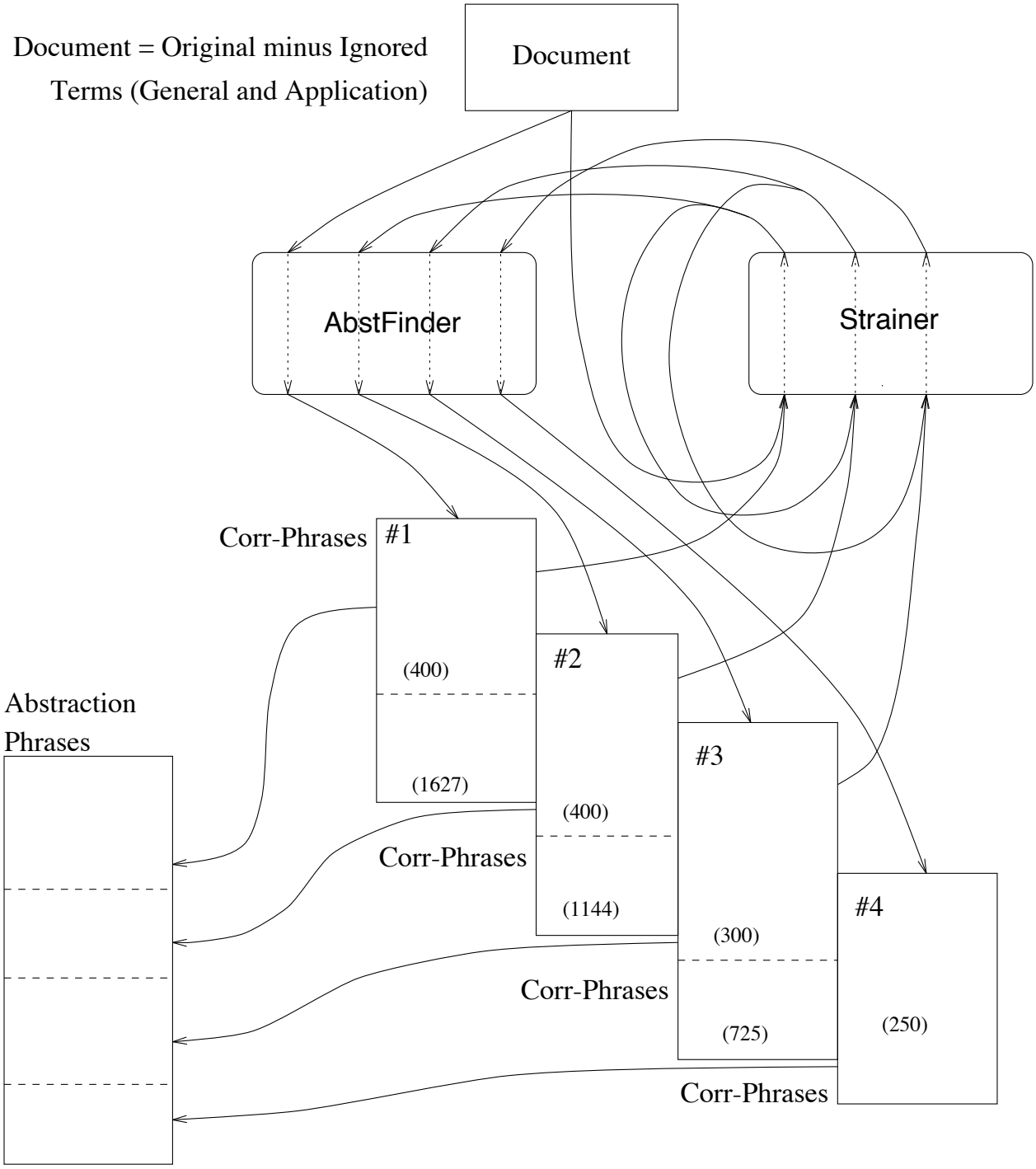


Figure 8: Iterative abstraction identification

ציור 8: זיהוי מופשטים באופן איטרטיבי

3.5 Combinations of Scenarios

The scenarios can be used in different combinations for different purposes. When using AbstFinder, the user has to be cognizant of the purpose or objective for the use, i.e., requirements, indexing, etc., and follow an appropriate combination of scenarios.

Let

1. "learning" denotes the activity of *learning what words to ignore*,

2. "well-organized" denotes the activity of *dealing with a well-organized document*,

3. "zooming" denotes the activity of *zooming*, and

4. "iteration" denotes the activity of *iteration to a final list of abstractions*.

Recall that all of these activity were described in previous sections (See Section 3). Thus, typical scenarios for requirements abstractions identification are the following:

[learning*] iteration, [iteration]

learning, iteration, [iteration*, learning*]

well-organized, learning, iteration, [learning*], [iteration], [zooming*]

The learning scenario is completly dependent on the human that does it. The human will decide to ignore terms, mostly application ignored words, according to his or her objective. For instance, when doing abstraction identification for requirements, many details, such as names of people, will not be considered as abstraction for requirements, and will be ignored during the learning scenario, by putting them in the *ignored-application-phrases-file*. On the other hand, when finding abstractions for the purpose of building a back-of-book index, details such as people's names are relevant candidates for indexing and will not be ignored.

In the indexing case study (See Section 4.3), it was demonstrated that two people with different objectives got different sets of identified abstraction from the same input, because they followed different scenarios.

Also, identifying abstractions in order to generate input for some requirements analysis method, such as OOA or SA will involve different scenarios according to the objective, which is identifying objects or functions.

4 Evaluation of AbstFinder

This section considers the evaluation of the effectiveness of AbstFinder for finding abstractions in natural language text. It is first necessary to explain how such a tool can be evaluated with the help of case studies. Then three case studies are described. These lead to the conclusion that for them, AbstFinder is indeed effective.

4.1 How to Evaluate a New Method or a Tool

With any new idea of a method or a tool one must evaluate its effectiveness. First, it is useful to compare the new tool to old tools, such as findphrases and the LA Finder, to verify that the new tool does at least as well or better than the old ones. One problem is that the old tools were tested against only toy examples.

One must really test such a tool against a human effort, since heretofore requirements elicitation has been done manually by a humans. There is no simple analytic method for testing human efforts. It is now well-known that controlled experiments do not work in software engineering. Because of the cost and the need to control the independent variables, controlled experiments are invariably on toy examples. Moreover, it is also well-known that conclusions do not scale up. The intellectual and managerial difficulties of a program grow exponentially with the size of the program. Running sufficient numbers of instances to obtain significant results is prohibitively expensive when the instances involve industrial-sized problems [Sch92]. It is just too expensive to contemplate repeating a real problem costing millions of dollars enough times to make statistically significant conclusions.

Moreover, too often, differences between individual software professionals dominate the controlled variable of the experiment [SEG68]. An early experiment that intended to compare two methods of program development produced non-significant results because a difference of up to 28 to 1 between pairs of programmers in their coding and debugging time and product size. An experienced programmer will almost always outperform an entry-level programmer. But Sackman worked with matched pairs of computer professionals, comparing, e.g., two individuals with 12 years of experience, and two entry-level programmers both trained in the same institute with same programming experience. What is most alarming about Sackman's results is that his biggest observed differences were between pairs of *experienced* programmers.

It is possible to consider repeating an experiment on a different project, rearranging the members of the teams, etc. The trouble with these techniques is that each experiment will probably last about a year, and there is no guarantee that all members of teams will stay in the same organization long enough to complete the experiment. Therefore the field of software engineering has been relegated to doing case studies with careful introspection

An important issue is the question of whether the abstractions found by the tool are meaningful to the human elicitor that has to approve them. Meaningfulness can be confirmed only by humans, and is very much affected by the *WordThreshold*. An abstraction identifier generated by AbstFinder is a sequence of characters called a phrase. This sequence of characters

was found independently of natural language word boundaries. Thus, it is important to check if these abstractions identifiers are meaningful.

If a word contained in a phrase that identifies an abstraction is not complete but at least contains the root, or if the abstraction is identified by more than one phrase, then the identifier is meaningful if a human can interpret the semantics of the phrases. For example, the following are meaningful identifiers: "`|solar radiation|`" is understood as is, and "`|surface |metal |`" means "metal surface", and "`|storage transit|`" means "storage transit", "storage transition" or "storage transiting", all of them can serve as an identifier describing the same abstraction. Whereas, "`|confi|identi|`" are meaningless, but when they appear together with "confidential", e.g. "`|confi|identi|confidential|`", then "confidential" is the semantic identifier that is extracted and the rest is noise and is neglected.

If we were sure that all chunks would be full and only words, then we could develop a mostly correct automated test for meaningfulness: submit the output word list to spell or another spelling checker. However, we are faced with phrases that do not stick to word boundaries, and meaningfulness can be confirmed only by humans, and is very much affected by the setting of *WordThreshold*.

Testing against human effort must show that the new tool does at least as well and possibly better than expert human elicitors in less time or with fewer people. Time and people power are easy to measure, but how to measure the concept of doing at least as well and possibly better than a human or other tool? A new tool should find at least all abstractions and maybe some not found by the humans or the old tool. Still, a criterion should be established for tools which involves comparing only input to expected output.

The key objective measures of the effectiveness of AbstFinder are: (1) its coverage, and (2) how summarizing it is. A tool that is not covering or which does not summarize is not good, for the following reasons:

It must be that this tool does not overlook any important abstraction that will need to be present in the requirements specification. A tool that does not overlook important abstraction is said to be *covering*. An elicitor will not be willing to be assisted by any tool unless he or she is confident that it is covering.

Clearly, the identity function is a covering tool. However, presenting all the input does not help the elicitor either. The other main requirement for the tool is that it reduce the amount of text that the elicitor must look at. An elicitor still has to do the *thinking* with the output of the tool, in order to approve the abstractions found. The elicitor will not be effective if the amount of information that must be examined is too big. A tool whose output is significantly smaller than its input is said to be *summarizing*.

Note finally, that a tool that is only summarizing is no good either. The most summarizing tool is that which outputs nothing. The tool must summarize while preserving coverage.

Measuring the ability to summarize is easy. It is done by simply comparing the ratio of sizes between the input transcript to the output of AbstFinder. Coverage is much harder to measure. One must compare the list generated by AbstFinder to that made by a known expert (and pray that in fact the expert is good) and judge whether all concepts found in the latter are present in the former. The high probability of error in this tedious job makes any claimed "yes" answer highly suspect. In addition, the person doing the job had a vested interest in finding a "yes" answer. Therefore a more systematic way to evaluate coverage had to be found.

This comparison can be done manually for small case studies, but it is almost impossible for a full-scale, industrial-strength case study. Thus, coverage question can be answered by straining from the human-made document all abstractions that appear in AbstFinder's result and seeing if there are any leftovers. This information in the leftovers has to be examined very carefully in order to find out if there are any meaningful concepts there. Lack of a meaningful concept means coverage is achieved. The smaller size of the leftovers and the greater visibility of meaningless text increases the credibility of the answer.

To put the evaluation in context, it is important to understand typical scenarios of abstraction identification with and without AbstFinder to help the elicitor. Without AbstFinder, the elicitor

1. reads all the documents once to get a sense of what is there, and

2. then repeatedly reads individual documents and parts thereof in order to find and verify abstractions until no more new abstractions are found.

With AbstFinder, the elicitor

1. reads all the documents once to get a sense of what is there, and

2. follows the iterative procedure of Section 3.4 until no new abstractions are found.

Thus, in traditional abstraction identification, the full set of documents are read over and over, with no prior limit. In AbstFinder-assisted abstraction identification, if the output is covering, the full set of documents is read only once. Thereafter, only the much smaller summaries need to be examined over and over. Besides being smaller than the full set, the summaries gather the most important concepts to the top of the list for a better focus.

For the evaluation, since the first steps of the two scenarios are the same, the comparisons focuses on the differences in what must be examined for the second steps.

4.1.1 The Evaluation Plan

On the basis of these evaluation criteria, the following evaluation plan was adopted.

1. Activate AbstFinder on the same problems used to evaluate findphrases and the LA finder in order to insure that AbstFinder is at least as covering as the old tools.

2. Activate AbstFinder on industrial strength case studies in order to compare its effectiveness to that of a human. This is done with two case studies, one smaller and the other larger.

    a. The Flinger Missile example is the smaller case. In this case study there was not any expert result to compare with. So, the author acted as an experienced elicitor, certainly not an expert, for evaluating the results of AbstFinder in this case study. The author may be biased, but the introspection gained was invaluable to learn methods to use AbstFinder and to prepare for the RFP case study.

    b. The RFP to IAI for the Unmanned Aerial Vehicle-Short Range system is the large industrial-strength case study. The RFP transcript was already analyzed by three expert IAI elicitors over a month. Their results were kept secret from the author while she was applying AbstFinder to the same RFP. The author was not an expert in the application area. Also, findphrases, a previous tool for abstraction identification, was run on the same RFP case study in order to compare AbstFinder performance to findphrases one.

The main purpose of the case studies is to compare abstractions found; if the elicitor assisted by AbstFinder finds all that the human experts found, then elicitors assisted by AbstFinder will be judged as at least as covering as only human elicitors. Another purpose is to see if elicitor assisted by AbstFinder found abstractions that the human experts did not find. There is no better measure than experience, and ultimately the proof will be in acceptance of tool by the elicitors' community.

Special attention will be given to the tuning of the *WordThreshold*, trying to get more meaningful abstractions with a higher *WordThreshold* while preserving coverage. Also, it is necessary to keep track of time and sizes of files while experimenting with AbstFinder on the RFP.

The experimentation with AbstFinder was conducted on a SUN4 server running a full load of users. However, it was not heavily loaded since the response to commands and of the editor were reasonable.

4.2 Case Studies

4.2.1 Findphrases Case Study.

The findphrases decomposition was used as a case study because the decomposition was already known. So, it could be used to check AbstFinder's results against already known results. The document that served as the requirements was the manual page of findphrases, because in fact the manual page was written as a requirements document, before the program was written. The requirements document was two pages long (See Appendix C.1). The already known abstractions were taken from Aguilera's [Agu87] program decomposition (See Appendix C.4), and her own list of abstractions identified by findphrases, and Maarek's [MB88] list of abstractions identified by lexical affinities (See Table 1).

The experience consisted of running once AbstFinder for learning purposes, in which the ignored files were initialized (See Appendix C.2). Then, one more activation of AbstFinder was needed for generating the abstraction list (See Appendix C.3). The following analyzes the results in terms of our criteria of evaluation.

Meaningfulness:

The output of AbstFinder applied to the findphrases manual page was found to be very meaningful. The correlated phrases that served as abstraction identifiers were complete from the linguistic point of view, and were very semantic terms, very similar to the real ones; "token file", "punctuation keyword file". The reader should do his own comparison with Table 1.

Summarizing:

The initial size of the findphrases transcript was 6935 bytes, whereas the AbstFinder output, via *corr-phrases-file*, was 1855 bytes, about 26% of the original data size. So, the output of AbstFinder is summarizing with respect to the original data.

Coverage:

As shown in Appendix C.3, the first 25 of the 48 entries of the AbstFinder output list includes all the abstractions found by Aguilera in implementing findphrases, all abstractions found by findphrases, and all abstractions found by Maarek with lexical affinities. So, for this case study, AbstFinder was found to be at least as covering as findphrases and the LA finder and was found to cover all abstractions found by a human programmer (See Table 1).

People and Computer Power:

The run of AbstFinder to generate the full output of the abstraction list took about 10 minutes. In this case study, the known abstractions of findphrases are the data abstractions used in the decomposition of the program. During the original decomposition of the findphrases program, no records were kept about the amount of time spent to arrive at the

The abstractions of findphrases		AbstFinder results	
Data Abstraction	Repeated phrases	AbstFinder's Corr-Phrases	Abst#
string_type_file	strings, characters	character|symbolcharacter|	7
argument_line	argument,option	argument loptional	3,14
output_file	output, tables of the output	output|tables	25
chunk_file	file(s), free format	free format |files	9
punc_keyword_table	punctuation keyword(s) file	punctuation keyword file	1
multi_tokens_table	multi tokens file	token file	5,17
text_file	text,input,arbitrary text	arbitrary text|input	12
phrases	phrase, repeated phrase, ignored phrase	phrase	4
sentences	sentence(s)	sentence	6

Table 1: The abstractions of findphrases
טבלה 1: ההפשטות של findphrases

decomposition. However, it is safe to say that decomposition took a lot more than 10 minutes of Aguilera's time.

4.2.2 Flinger Missile Case Study

The document for the Flinger Missile case study was five pages long and contained 279 sentences. Its first part, 65 sentences over 1.5 single-spaced pages, contained a well-defined functional specification of the Flinger Missile (See Appendix D.1). That part was well written in a very professional language. Its second part, 214 sentences over 3.5 single-spaced pages, contained the transcript of a video-taped unrehearsed interview between the elicitor and an operator of the Flinger Missile, in which the elicitor was questioning the operator about the missile behavior.

Since there was no human-made abstraction list to which to compare the AbstFinder results, the author had to play both roles:

1. The elicitor assisted by AbstFinder working on the Flinger Missile document, called elicitor-A below.

2. The expert elicitor, called elicitor-X below, confirming that the identified abstraction by AbstFinder are real abstractions in the Flinger Missile document.

The author was not acquainted at all with the Flinger Missile project, but did have good experience in developing real-time software for computer-based systems [WAHKMOOTW93, ALW91]. Thus, the author combined both roles: activating AbstFinder on the document as elicitor-A, then following AbstFinder output as elicitor-X, confirming that each abstraction identifier in that list actually defines a real abstraction in the original document. This conclusion may be biased, but the experience gained with this case study helped to learn how to use AbstFinder. However, note that elicitor-A had become more expert with the application by the time she had to switch to being elicitor-X!

The following is a description of the experiment steps

1. First, AbstFinder was run a few times on small parts of the Flinger Missile document, taken from different places in the transcript, in order to initialize the ignored phrases and suffixes files (See Appendix D.2).

2. AbstFinder was activated on the first part of the Flinger Missile document (See Appendix D.3).

3. AbstFinder was activated on the whole document.

The following analyzes the result in terms of our criteria of evaluation.

Meaningfulness:

AbstFinder on the first part of Flinger Missile transcript produced very meaningful abstractions. For instance, “launch”, “circle”, “orientation”, “self test”, “cruising speed”, “artificial horizon”, “loiter”, “reconnaissance”, “friend foe target”, “self destruct”, etc., are very essential functions and objects of the real-time system of a missile. The first part of the Flinger Missile document was written in a very professional language, which explains the very professional meaningful list of abstractions identified by AbstFinder. Activating AbstFinder on the whole document, including the interview, did not reveal any new information. Of course, the interview part of the document could not stand by itself, because its questions and answers did not cover all the details of the system.

An interesting thing occurred to elicitor-X while verifying the abstractions of Flinger Missile found by AbstFinder. Two abstractions were found by AbstFinder, “loitering” and “reconnaissance” whose functional difference was not clear at first glance. In general, loitering is the unemployed state while reconnaissance is the spying state. This makes a lot of sense when trying to identify the missile’s different modes. Loitering is a default mode, in which the camera does not operate, while reconnaissance is an active mode, in which pictures are taken. The nice thing was that in the interview part of the document, the real elicitor mentioned in the document was also confused about those two different modes of the missile, and was asking the operator the same question. So the team consisting of the author and AbstFinder did make a good elicitor.

Coverage:

Since there was no reference for comparing AbstFinder results to human elicitor efforts, it was impossible to check coverage as defined. Still, a rough estimation of coverage was obtained using Strainer (See Section 3). Strainer was activated on the Flinger Missile document, removing from the original document words taken from *corr-phrases-file*, e.g. the abstraction identifiers found by AbstFinder, including terms found in the *ignored-application-file*. Recall that *corr-phrases-file* contains only the abstraction identifiers in terms of the original transcript. These abstraction identifiers are the only information that the elicitor needs to scan and all the other parts of the AbstFinder output is primarily for research purposes and not intended for the elicitor.

The leftovers of the original document, after straining out *corr-phrases-file* words, were checked very carefully by elicitor-X and found to be meaningless (See Appendix D.4). This helped to gain confidence that AbstFinder found all the terms that are essential for understanding the Flinger Missile document. Without the terms that AbstFinder identified as abstractions the text is meaningless, meaning that no concept or abstraction was left and the abstraction list of AbstFinder does cover the important issues of Flinger Missile.

Summarizing:

The initial size of the Flinger Missile transcript was 15029 bytes, while the AbstFinder output, via *corr-phrases-file*, was 3429 bytes, about 22% of the original data size. So, the output of AbstFinder is summarizing.

People and Computer Power:

The run of AbstFinder on Flinger missile took about an hour, including the analysis. Again, there were no human efforts to compare with. Still, an hour for generating a list of abstractions is minor in terms of human effort and performance, and it is clear that the original effort took more. To sum up, the introspection gained from the case study was necessary to learn methods of using the tool and to prepare for the RFP case study. For instance, only by experimentation, it was found that a *WordThreshold* of 6 is more reasonable than of 5, as had been anticipated. Also, clearing all noise is not so good. Some reasonable amount of noise helps to differentiate better between the abstractions that are found. Also, using Strainer for estimating coverage led eventually to the scenario of using AbstFinder iteratively.

4.2.3 RFP Case Study

The Request For Proposal (RFP) document for the Unmanned Aerial Vehicle-Short Range (UAV-SR) system [IAI89] is a real life case study, about 100 pages long, containing 2200 sentences (Appendix E.1 shows some portions of the RFP that help the reader to see the basis for the following analysis). The Israeli Aircraft Industry (IAI) is currently producing the UAV-SR according to the client's needs expressed in its RFP document. There are two groups in IAI that work in parallel, and meet only in pre-planned rendezvous points during the life cycle of the

project (See Figure 9). One group is the development group, which carries the product through the different life cycle stages; analysis, design, etc., using the RFP as the basis document that defines the user's needs. The second group is the Software Quality Assurance (SQA) group, which is responsible for doing reviews at different pre-planned points in the life cycle of the project, in order to control the software development process.

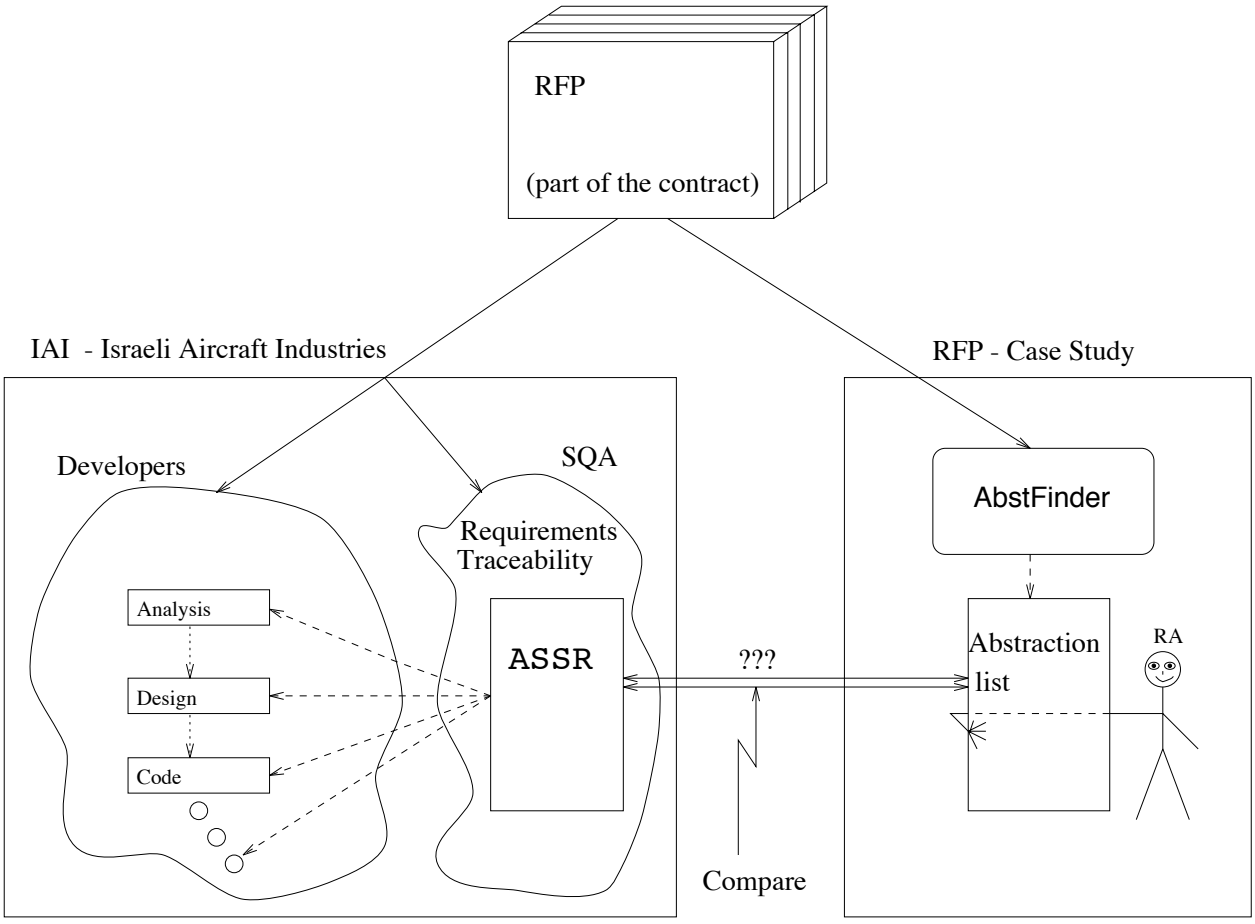


Figure 9: An illustration of the RFP case study
ציור 9: איור של דוגמת לימוד RFP

The SQA group's main interest now is to solve the problem of requirements traceability. Requirements traceability consist of finding for each requirement the module or modules which help meet it. For this purpose, they had captured the requirements in the user's and client's language taken from the RFP, and expressed them as a list of simple statements, each identifying one requirement that is written in another document called the Allocated Software System Requirements (ASSR) [IAI90]. The IAI people did a simple grammatical transformation of complex sentences into simple sentences, each containing one requirement in a strict format, for example, "The software shall provide the capability to support the AV with guidance and control" (Appendix E.5 shows some portions of the ASSR that help the reader to see the basis for

the following analysis). Notice that "The software shall provide the capability to" is a fixed pattern used by IAI for stating a requirement, in order to comply with the MIL-STD 490 [DOD85] for writing specifications. Clearly, words from this and other similar phrases should be in the *ignored-application-phrases-file*.

For creating the ASSR, IAI employed three people for a month. They read the user's document over and over and extracted from it the simple requirements statements. Then, another three people had to approve it. The final document that the IAI people produced contained all the client's requirements known by the IAI people and the developers. The list of requirements produced by the three experts is called the "human-made" document below, and it served as a reference list for the experiment. This project was already done by the IAI. So the requirements were well-known to them, but they were kept secret from the author.

The experiment consisted of the author, called the elicitor below, using AbstFinder on the RFP to generate the list of abstractions, and comparing the resulting abstractions to the IAI's human-made list of requirements. The elicitor did not see the human-made document until after finishing to generate her output list of abstractions. The hope was that the elicitor would find *meaningful* abstractions in a *summarizing* output list of AbstFinder while providing full *coverage* of the client's requirements, and to get all that with a lot less effort than three person months.

The elicitor (the author) had never worked in any kind of application similar to that described by the RFP document. Even the title of the RFP document, *The Unmanned Aerial Vehicle-Short Range (UAV-SR) system*, did not tell her anything as an elicitor. So the elicitor had to dig into the RFP with the help of AbstFinder in order to identify the abstractions of that RFP. A copy of portions of the RFP document is included in Appendix E.1 to allow the reader to follow the description below.

The experiment involved five steps.

1. First, some trial runs were done on small parts of the transcript, in order to learn the language of the document and initialize the ignored files (See Appendix E.2).

2. The RFP was a well-organized document, being well structured with a table of contents and divided to sections and subsections. The first run of AbstFinder was done on the original transcript as is, and there was a strong resemblance between the leading abstractions in the list and the table of contents of the RFP document. Some of the abstractions belonged to the meta-language of document writing. These included "operational organization concepts", "design construction", "summary", "confidential", and "base-line configuration". These were added to the *ignored-application-phrases-file*.

3. For the next test, the table of contents was removed and AbstFinder was activated again on the RFP with the updated files of ignored words. The ignored words and suffixes files were updated after each run, and they converged after three or four runs. The results were very meaningful, and neither concepts from the table of contents nor the meta-language of document writing appeared. The initial output list of AbstFinder was very

long and contained essential objects and functions of an air vehicle, such as, "launch recovery", "mission", "test", "position", "transponder", "acceleration", etc. (See Appendix E.3). It contained 1627 abstractions (with repetitions). The elicitor observed the abstractions consisting of less than four phrases, and stopped after about 400 abstractions in AbstFinder output list. These 400 were logged to the abstraction phrases file that accumulated the final result of AbstFinder.

4. The elicitor decided to zoom (See Section 3) into abstractions that were found to be too broad. For instance, "testing" was a very good abstraction, but it was too broad and unfocussed and the elicitor decided to be more specific. The result of that zooming was, for instance, a detailed list of sub-abstractions of the abstraction "testing" (See Appendix E.6). So, a better resolution of all the concepts that relate to "test" was achieved: "test equipment", "acceptance tests", "rejection/retest", "test levels", "burn test", "environmental tests", "test flight phases", etc. The same was done for "fail" and "safety" (See Appendix E.7).

5. Following the iterative process (See Section 3), Strainer was activated in order to remove from the input transcript of this stage the abstractions that were already recognized. Again, the elicitor read the list of abstractions found in, up to about 400 abstractions, which were identified by less than five phrases, and stopped again because of readability problem. After three times of applying AbstFinder and then Strainer, the elicitor was finally left with a very short list, of only about 250 abstractions, which were all meaningless. Most of them were parts of words such as "aiming", "partic", "rable", etc. (See Appendix E.4). That only meaningless abstractions remained indicates that all the meaningful abstractions were identified previously and strained out from the transcript. The accumulated list of all the meaningful abstractions, that were strained out during the iterative process, is the final output list of the AbstFinder effort.

At this stage, the elicitor met the IAI people and exchanged documents; the elicitor gave them the AbstFinder output list and they gave her the heretofore, secret ASSR document [IAI90]. The following analyzes the results in terms of our criteria of evaluation.

Meaningfulness:

After the elicitor was finished generating what she thought was a complete list of abstractions, the phrases in this list were examined by three expert analysts of the RFP transcript. The three professional requirements analysts, Mr. Kudish, Dr. Winokour, and Mr. Engel, are highly skilled and have nearly sixty years of cumulative experience in real-time system and software requirements analysis. They all said that they found all of theAbstFinder-generated phrases to be meaningful to them. They confirmed that the abstraction identifiers generated by AbstFinder contained terms and phrases that identify real functions and objects of the RFP that they already knew. One of the IAI people, Dr. Michael Winokur, was very impressed to see in the beginning of the AbstFinder-generated list some abstractions, such as "surrogated training", that they had overlooked for a long time until finally the customer pinned it on their noses.

Summarizing:

The output of AbstFinder was summarizing. The original document RFP was 214,654 bytes long while the final AbstFinder-assisted list was only 47,105 bytes, about 21% of the size of the original data. The original transcript contained full sentences of text. The AbstFinder result, via *corr-phrases-file*, contained only the abstraction identifiers, one per line, that were recognized and logged by the elicitor. Recall that AbstFinder output contains noise of parts of words, and repetitions, both between abstractions and within an abstraction.

Since the elicitor uses the AbstFinder abstraction list as a guideline for understanding the original document, it is important that it be small enough and contain the gist of the material in the original. In fact, the experts observed that the output of AbstFinder, *corr-phrases-file*, could serve as input for generating the ASSR. They felt that the abstraction identifiers together with the interpretation of the elicitor, subjected to the strict format of the ASSR would yield the ASSR statements. For instance, "launch recovery" would yield "The software shall provide the capability to do launch recovery".

Coverage:

The problem with evaluating coverage is that someone must sit down and see that all abstractions in the human-made document show up in the AbstFinder-generated list. The high probability of error in this tedious job makes any claimed "yes" answer highly suspect. In addition, the person doing the job has a vested interest in finding a "yes" answer. Therefore, a more systematic way to evaluate coverage had to be found.

The human-made document, the ASSR (See Appendix E.5), contained a list of requirements statements in a very strict format, while the AbstFinder output is a list of phrases, in free format, identifying abstractions. Recall that an abstraction is not a requirement, but they are both stated in natural language words. So, by removing from the ASSR all words that identified abstractions in the AbstFinder gives a good measure of coverage. The leftovers of this subtraction above contains natural language words that appeared in the human-made document and did not show up in AbstFinder result. If some meaningful concept can be find among those words, then AbstFinder has failed to cover all the abstractions-concepts in the RFP document.

The coverage question can be answered by straining from the human-made ASSR document all abstractions that appear in AbstFinder's result and seeing if there are any leftovers. No leftovers means full coverage. The smaller size of the leftovers and the greater visibility of meaningless text increases the credibility of the answer.

Strainer was used again for removing from ASSR words taken from AbstFinder output via *corr-phrases-file*. Since *corr-phrases-file* is as generated by AbstFinder, the list contains abstractions that are identified by more than one phrase with repetitions and parts of words. For instance, if an abstraction is identified by two concepts a and b, it will appear as $a \mid b$ in AbstFinder output list. The following explains why this format of AbstFinder output does not harm the coverage verification process. If an abstraction is identified by $a \mid b$, then there are some

possibilities:

1. If *a* contains the same word as *b* with extra blanks on one or both sides, then Strainer removes it only once, because it works with whole words only. Strainer takes a word from *corr-phrases-file*, with white space on both sides, and removes the word from ASSR only if it found the word there with white space on both sides. So, if *corr-phrases-file* contains "`|launch|`", or "`| launch|`", or "`|launch |`", or "`| launch |`", Strainer will remove "`launch`" from ASSR, if it appears therein surrounded by white spaces.

2. If *a* is substring of *b* or vice versa, then Strainer, which is working only with whole words, ignores it.

3. If *a* and *b* are two different phrases then:

    a. If *a* is meaningful and *b* is a common word, e.g., among the sub-abstractions of "testing", "`|accordance |test levels |`", then Strainer will remove also the common word, i.e. "accordance" that is actually a noise and should have been added to the ignored files.

    b. If *a* and *b* are both meaningful abstractions, then the elicitor will accept both of them as two different abstractions.

The result of the subtraction of *corr-phrases-file* from ASSR was 3019 bytes. The RFP was 214,654 bytes (about 100 pages) long and the human-made requirements document was 83 pages (about 140K bytes) long. The phrases of the remainder were analyzed very carefully in order to see if AbstFinder missed any abstraction. The phrases of the remainder were separated to several categories according to their characteristics.

1. Most of the phrases originate from the strict meta-language of the requirements specification format of the ASSR, such as "activate", "allow", "deactivate", "herein", "include", "integrate", "must", "only", "provide", which are not abstractions and were used only in the ASSR document for stating requirements and not in the RFP original transcript.

2. Some concept were in different grammatical forms such as "transmit" in the AbstFinder abstraction list, and "transmitting", "transmitters", and "transceiver" in the ASSR. Those words in the leftovers do not carry any new concept, they actually describe the same abstraction. The same is for:

    "calibrate" and "calibration"
    "assigned" and "assignment".

    While AbstFinder is designed to classify all the "transmit..." words as a single abstractions, Strainer is designed to remove only whole words and does not remove words that properly contain a recognized root. If Strainer were to cut parts of words, then the

remainder of the document will be a mass of unreadable text. For instance, suppose that "inter" were found by AbstFinder as a common part among "interchangeability" and "interfaces". Removing "inter" as part of word would leave in the leftovers "changeability" and "faces". Both of these accidentally generated words are garbage relative to the application.

3. Acronyms such as "NBC" are introduced to replace a longer full phrase such as "Nuclear, Biological, Chemical"; the full phrase appears only once at the introduction of the acronym or in a dictionary of acronyms, and the acronym appears many times throughout the document. The acronyms are used to save the writing of the longer full phrase. AbstFinder did not identify many acronyms. Many acronyms are shorter than the *WordThreshold*, and a full phrase if appears only once it is not going to be caught by any frequency-based scheme. Actually, only the "NBC" was not found, all the others were found since the term of the acronyms were repeated in the text more than once. Given that reducing *WordThreshold* causes generation of too much noise, there are two solutions, both general enough to be made part of a standard scenario for the elicitor.

    a. The synonym dictionary can be used to replace the acronyms by their full phrases for the purpose of abstraction identification.

    b. Recognize all the acronyms as important abstractions, log them as abstractions, and then add them to the *ignored-application-phrases-file*.

    Only after recognizing the abstractions, the elicitor may switch to using acronyms as abstractions identifiers.

4. Ten concepts appeared in the leftovers because they appear in the RFP only once, and AbstFinder identifies only concepts that appear more than once, at least once for definition and once for use. Of these, five phrases were synonyms in the context of the system that was defined in the RFP, such as "contour" and "elevation", and "enemy" and "threats", that occurred because the *synonym-file* was not implemented yet.

5. The remaining five phrases were specific examples of some already captured abstractions and appeared in the text with linguistic clues, "i.e.", "e.g.", and "for example". These are not abstractions, they are details that will be put inside the abstraction.

To sum up, after some generally applicable modifications that should be part of a standard scenario for use of AbstFinder, full coverage was achieved.

Does Better than Human Experts:

It was interesting to see if AbstFinder found some concepts that the human elicitor overlooked. This meant checking if the list of requirements in the human-made document cover the list of abstractions found by AbstFinder. That question was answered by removing from the AbstFinder-generated abstraction list all that appear in the human-made document to see if there are any leftovers. Again, the subtraction was be done by Strainer.

The result was about 35,402 bytes long. There were very meaningful concepts concerning "communications", "ordnance", "weather conditions", etc. Perhaps some of these did not appear in the human-made document because they were hidden in the classified requirements appendix of the RFP document. This appendix is competition sensitive and was not submitted to the research case study. Note, that the RFP specifies the whole system, hardware and software, while the human-made document specifies the software only. So, most of these 35,402 bytes concern other requirements than software. A great portion of these leftovers was noise, i.e., parts of words, and did not contribute any new concept.

We also found that the people of the project were not happy to hear the results of this investigation, because the project was already in progress, and they felt, incorrectly, that it was not the right time for them to find things that they might have missed. Note again, that according to the project people, some of the abstractions such as "surrogated training" appeared very clearly at the output list of AbstFinder while the project people overlooked it for long time. So, we got the impression that an elicitor operating AbstFinder can do better than a group of human elicitors.

People and Computer Power:

The list of requirements generated by the three experts required one month of concentrated work for a total of three person-months. Running AbstFinder took about five hours total CPU time, three hours operating time, and about two hours of elicitor overview, which is about one day of work. The first run of AbstFinder on RFP took about two hours. The second run, after straining out the most frequent abstractions on the list, took about 30 minutes. The last run took about 5 minutes. However, note that the elicitor was doing other things while the CPU was running.

4.2.3.1 findphrases vs. AbstFinder

In order to demonstrate the improvement of AbstFinder over old tools of abstraction identification, an experiment was conducted in which findphrases was activated on the RFP source document that was used as the source for the RFP case study. The main concern was to check the findphrases results of abstractions identification, in terms of *coverage* and *meaningfulness*. findphrases was run using the same ignored words obtained from the common and application ignored words files. Since findphrases treats the ignored words differently, an ignored word does show up when it is used in conjunction with some non-ignored word.

The output of findphrases contained a much smaller number of identified abstractions, which immediately implied that would be a coverage problem (See Appendix E.8). An examination of the findphrases results revealed the following:

a. findphrases found most of the major concepts consisting of one word, but still failed to find a lot of one word concepts, such as, "transponder", "generators", "navigation", "acceleration", "telemetry", "logistics", etc. which are very important application functions and objects. Probably those concepts appeared more than once in the transcript of

RFP since AbstFinder found them, but their number of appearance was very low relative to the frequent ones. Thus, they did not pass the minimum frequency factor and were not shown in the findphrases output.

b. findphrases completely failed to find compound phrases of two or more words that identify an abstraction. For instance, the abstraction "remote video terminal rvt", which is a very important object in this application, was found very clearly by AbstFinder. findphrases, however, found only parts of the term and they were spread all over the list, such as, "remote", "video", "terminal", "data terminal", "terminal (". These fragments do not imply the full concept at all. All of this is the result of findphrases's approach of looking for fixed patterns. On the other hand, AbstFinder is completely flexible in finding a term in any permutation of its words and with any gap between them. The same happened to "real time"; findphrases found only "real" and "time". The phrase "ground data terminal gdt" was not found by findphrases either. Instead, even though it was looking for phrases of length up to five, findphrases found only "ground", "ground data", "data terminal", and "terminal", again spread all over the list. findphrases failed completely to identify "baseline configuration" and "functional audit", which are very important quality requirements.

c. Actually, phrases of more than one word were rarely find in the output of findphrases on RFP. Usually, if a phrase of more than one word was found, then the adjacent word was a common one, such as, "the system", "equipment shall", "the equipment", "the system shall", "contractor shall", etc. These phrases do not add more information. Again, this limitation is the result of the approach of fixed patterns. All of the above confirm the observation that the abstractions found by findphrases do not cover all the abstractions that exist in the RFP case study.

When comparing the output of AbstFinder to findphrases, the AbstFinder output has much more meaningful compound abstractions that describe the system to be built, and help in understanding the requirements. In that context, the findphrases output looks like a list of single words, without context, ordered according to their frequency in the source text. In terms of meaningfulness, AbstFinder is much better.

4.2.4 Results

For the specific case studies carried out, the author assisted with AbstFinder was found to be

1. at least as good as findphrases and the LA finder on the findphrases requirements. All the abstractions found by findphrases the LA finder were found at the top of the output list of AbstFinder.

2. at least as good as the author acting as an elicitor on the Flinger Missile example. Through this case study, the Strainer was found to very useful for estimating coverage. This case study was good practice for learning scenarios for using AbstFinder.

3. at least as good as three human experts on the RFP, and in fact found some abstractions that they did not found. For the same case study, the author assisted with findphrases failed to cover all the abstractions and yielded less meaningful abstractions identifiers.

In all the case studied, the AbstFinder output was summarizing with respect to the original document (See Table 2).

	findphrases	Flinger Missile	RFP
Source-file	6935 bytes	15029 bytes	214654 bytes
corr-phrases	1855 bytes	3429 bytes	47105 bytes
%	26%	22%	21%

Table 2: The Case Studies Summarizing Criteria
טבלה 2: קריטריון התימצות בדוגמאות הלימוד

Moreover, in the RFP case study, the AbstFinder output amounted to 21% of its input, and all the runs of AbstFinder on the RFP to determine its abstractions took one day, while the three human experts took three months to do their analysis of the RFP.

The conclusion is that for the case studies presented, AbstFinder output is meaningful, has coverage, and is summarizing.

Recall that AbstFinder output contains noise strings and repetitions. While experimenting with AbstFinder, the author found that some noise was helpful for differencing the identified abstractions, although it raises the output size. Additional filtering of repetitions and noise would reduce the summarization factor substantially (See Section 4.2.3).

More experiments on industrial sized examples must be carried out. With each such experiment, it is important to have a qualified, independent analysis available with which to compare the AbstFinder-generated list of abstractions.

Also now that the prototype has successfully proved a concept, it is time to consider scrapping the oft-modified prototype in favor of a freshly written production version, in which better algorithms and data structures are used.

4.3 Indexing vs. Abstraction Finding

Some consider abstraction identification and term identification for indexing (as for the back-of-a-book index) to be similar activities and believe that a tool for one should be usable for the other. To get some idea of how true these thoughts are, we conducted an experiment in which AbstFinder was used to help find indexing terms.

criterion	findphrases	Flinger Missile	RFP
meaningful	very	very	very
summarizing	26%	22%	21%
coverage	o.k.	o.k.	o.k.
do-better	configuration files	?	surrogated training
performance	15 minutes	45 minutes	3 hours
Source	6935 bytes	15029 bytes	214654 bytes

Table 3: Case Studies comparison
טבלה 3: השוואת דוגמאות הלימוד

There exists a published paper about tool-assisted term identification and a ditroff post-processor for generating the index based on a file containing the terms to be indexed. This paper has an index generated by the tools described therein [AB89]. Interestingly, Berry, the advisor of the author of this thesis, is a co-author of this paper. It was decided to have both the author of this thesis and Berry. The author of this thesis is, of course an expert AbstFinder user, but has never done indexing. Berry, of course is very familiar with the paper, has participated in its indexing, but had never used AbstFinder before. This experiment also served as a way to test the distributed version of AbstFinder, consisting of binary programs, shell scripts, and manual pages, for usability by someone other than the author. Indeed, Berry ended up suggesting a number of minor improvement to the shell scripts and manual pages.

Berry used two iterations to get AbstFinder to generate about the same set of terms that had been identified automatically by the phrase finder tool used for the paper. Actually AbstFinder was a little better than phrase finder for this purpose, but that is no surprise. However, still only about half the eventual list of indexing terms were identified automatically. This result was not surprising to Berry because he already knew the term identification and abstraction identification are significantly different processes. Indeed, in one sense they are opposites. Consider a phrase appearing only once. It certainly does not constitute an abstraction that encapsulates data and algorithms. Yet the very fact that it appears only once is a strong reason for indexing it to allow someone who would miss the concept because it is not listed in the table of contents to find it. In this respect, the table of contents can be thought of as a list of abstractions, while the index can be thought of a list of abstractions *and* of specific details. This observation leads to the conclusion that a successful abstraction finder is a poor automatic index term identification tool and probably vice versa. AbstFinder is just not a good index term identification tool and is not expected to be.

The author of this thesis, in her runs of AbstFinder, seems to have forgotten that the goal of the exercise was to generate index terms, and ended up generating a list of abstractions. This information, however, cannot be compared with anything as no list of abstractions was identified earlier and published. Interestingly, Berry observed that these abstractions could form a basis

for organizing the paper. The names of the abstractions would become the section titles and would show up in the resulting table of contents.

5 Abstraction Organization Exploration

The abstraction identification process, the issue dealt in the previous sections, results the abstraction identifiers which are the important concepts in the user transcript, but are not enough for elicitation. Once the abstractions are recognized, they have to be organized on some mean so that all the text dealing with one abstraction can easily be found. Only then, the elicitor will be able to do the elicitation, i.e., ruthlessly filter out inconsistencies, identifying absence of information, negotiating with the customer-user, and updating the abstraction content if necessary.

The objective of the following is to explore the use of hypertext for abstraction organization for the purpose of determining what operations will be needed, both to be done automatically by the hypertext and to be available by the elicitor. This exploration is limited to identifying needed functionality and is not for evaluating feasibility. A lot of research has been done recently, as will be shown later, in hypertext itself and in its use for software engineering. Thus, this exploration intend to present a bridge between the phase of identified abstractions to this other work. As will be shown later, all of this hypertext work assumes that the information is already segmented to abstractions by the user of the hypertext.

5.1 What is an Abstraction Network

The goal of REGAE, as mentioned in Section 1.5, is to help the elicitor build an abstraction network. The elicitor is to use this structure to analyze the entire information in an associative way. The elicitor is to be able to navigate through the structure to get more details about any abstraction. Each node will contain the information about an abstraction collected from several original documents of requirements description, so that it will be easier for the elicitor to identify conflicts or complementary information about an abstraction taken from different user interviews. Furthermore, this organization gives the elicitor the ability to identify mutual influence among abstractions and to add his or her own requirements based on that influence.

During the first stage of operating REGAE the abstractions are identified, by the AbstFinder algorithm described at Section 2.3.5. The output of this stage is a list of abstractions identifiers. Each abstraction is identified by a phrase, which is the one that caused the similarity between the sentences of that abstraction.

The second stage of operating REGAE is to use the output of the first stage, the identified abstractions, together with the original interviews and to build a network, with the aid of a software tool which automates in a nonobtrusive way as much as usefully possible. The output of the abstraction identification process is not sufficient in order to proceed to the next stage. It is necessary to involve the elicitor in refining the abstractions found by the AbstFinder algorithm and getting his or her approval before proceeding to the next stage. Indeed, any requirement to involve humans should not be looked on as a drawback, rather it is an opportunity for a human to think and be creative. The purpose of automating what is possible is to free the human from clerical work to do creative thinking.

The goal is to build a network similar to that shown in Figure 10. In such a network, there are two kind of nodes; *text nodes* and *abstraction nodes*. A text node contains an original document of user interview. A long document may be cut into several text nodes connected by *text links*. The original transcripts must remain unchanged, sequential, in order to be used as reference and see concepts in their original context. An abstraction node embodies an abstraction. An abstraction node contains the sentences of the abstraction, which were found to be similar. The phrase that caused this similarity between those sentences is used to identify the abstraction. Abstraction nodes are connected by *Abstraction links*, which are generated automatically by the phrases that identify the abstractions, as will be explained later. *Data links* (pointers) to the original text located in text nodes, should be added. The data links are used to connect sentences in the abstraction to their original transcript, in case one might want to look at a sentence in its original context. All links are double ended, in order to allow bi-directional navigation.

As suggested in Section 1.5, this network should be built on top of some existing hypertext system.

The process of generating the abstraction network is as follows:

1. Build the text nodes for each user interview document.

2. Build an abstraction node for each abstraction by automatically collecting all the sentences that belong to that abstraction, according to keywords contained in the phrase that identify that abstraction.

3. For each abstraction, automatically construct abstraction links according to the keyword of phrases identifying other abstractions.

Several operations are done with the sentences in the text nodes. During the second step above, the sentences in text nodes which are pointed to by data links are marked, and for each of these sentences, a trace is kept backwards to the abstraction nodes that are the origins of the data links. The problematic sentences in text nodes which are not linked remain unmarked, so that the elicitor can check their relevance.

However, one problem remains. There may still be sentences in some text nodes which are not linked with any abstraction nodes. Usually, these sentences have meaningless information, as often happens in natural language transcripts, e.g., “We now turn to specific examples of these requirements.” Sometimes, there are sentences that seem to contain relevant information, that is context dependent, such as those beginning with “Consequently, it ...”. In any case, all these unmarked sentences are gathered into a some abstraction called *garbage-collector node*, for the elicitor to analyze it.

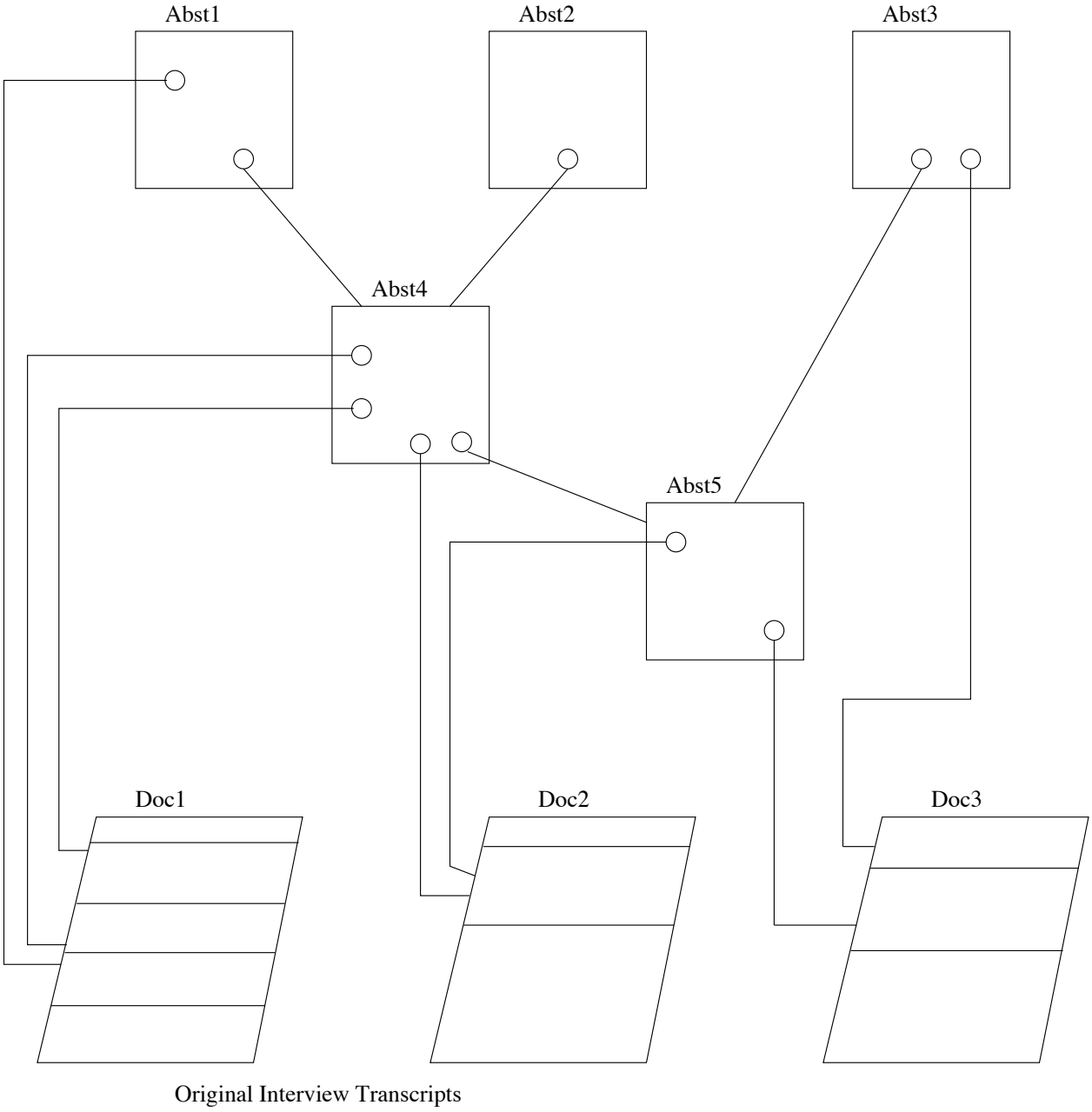


Figure 10: The abstraction network
ציור 10: רשת של הפשטות

5.1.1 Using the Abstraction Network—The Perspective of the Elicitor

At this stage, the abstraction network is ready for a skilled elicitor. The elicitor would ruthlessly filter out inconsistencies between sentences not only within a node, but between nodes as well. The elicitor would use links to navigate between the nodes in order to track down these inconsistencies by verifying information from different abstractions (nodes). Any failure by the elicitor to find a link where he or she needs one is an indication of a place in which links should be generated automatically, or an indication of a link that the elicitor should generate. The key question here is whether there are more mundane, clerical parts of this activity that can be automated.

Recall that an abstraction is not equal to a requirement, but abstractions can serve as an initial list for requirements, and can be used for the negotiations with the customer. By looking into the abstraction content, the elicitor may ask the customer if that is all that is known about that specific abstraction. This content may contain inconsistencies, or may be too small and vague. In both cases, the customer is invited to resolve any problem with the abstraction content [LF91]. Only then, The abstraction can be rephrased as a requirement or several requirements.

Once the requirements are written, it is necessary to validate these requirements for compliance with what the client desires. It is currently envisioned that the elicitor would invite the interviewed members of the client organization to read the draft requirements, and they would evaluate the compliance of the document with their conceptions.

The elicitor can analyze the requirements description, and may want to make some changes, corrections, deletions, or additions. It is very important to keep the original data, in the text nodes, unchanged. Therefore, a new node type is defined, called *issue node*, and appropriate links. The *issue links* connect abstraction nodes to changes written in issue nodes. The elicitor can use these issue nodes in order to make corrections or additions.

The issue nodes and links can be arranged according to the Issue-Based Information Systems (IBIS) method [Con87, CB88]. The IBIS method is based on the principle that the design process of complex problems, termed wicked problems, is fundamentally a conversation among the stakeholders, e.g., designers, customers, implementors.

In a way similar to IBIS, it is proposed to use other kinds of nodes, such as issues, positions, and arguments, and other kinds of links, such as responds-to, questions, supports, objects-to, generalizes, refers-to, and replaces. In a typical application, requirements documents are presented to the client and users, and then the elicitor begins negotiations with them. These people involved express their positions about an issue the elicitor brings up, and give arguments to their position. All of these positions and arguments may be entered into the same abstraction network. The elicitor reads the original information, and writes his or her analysis comments into the issue node. The participants express their positions using argument nodes and position nodes together with appropriate links. Each of these nodes may become a seed for a new conversation. See Figure 11. This organization is a good representation for the people that have a stake in the project [BYC90].

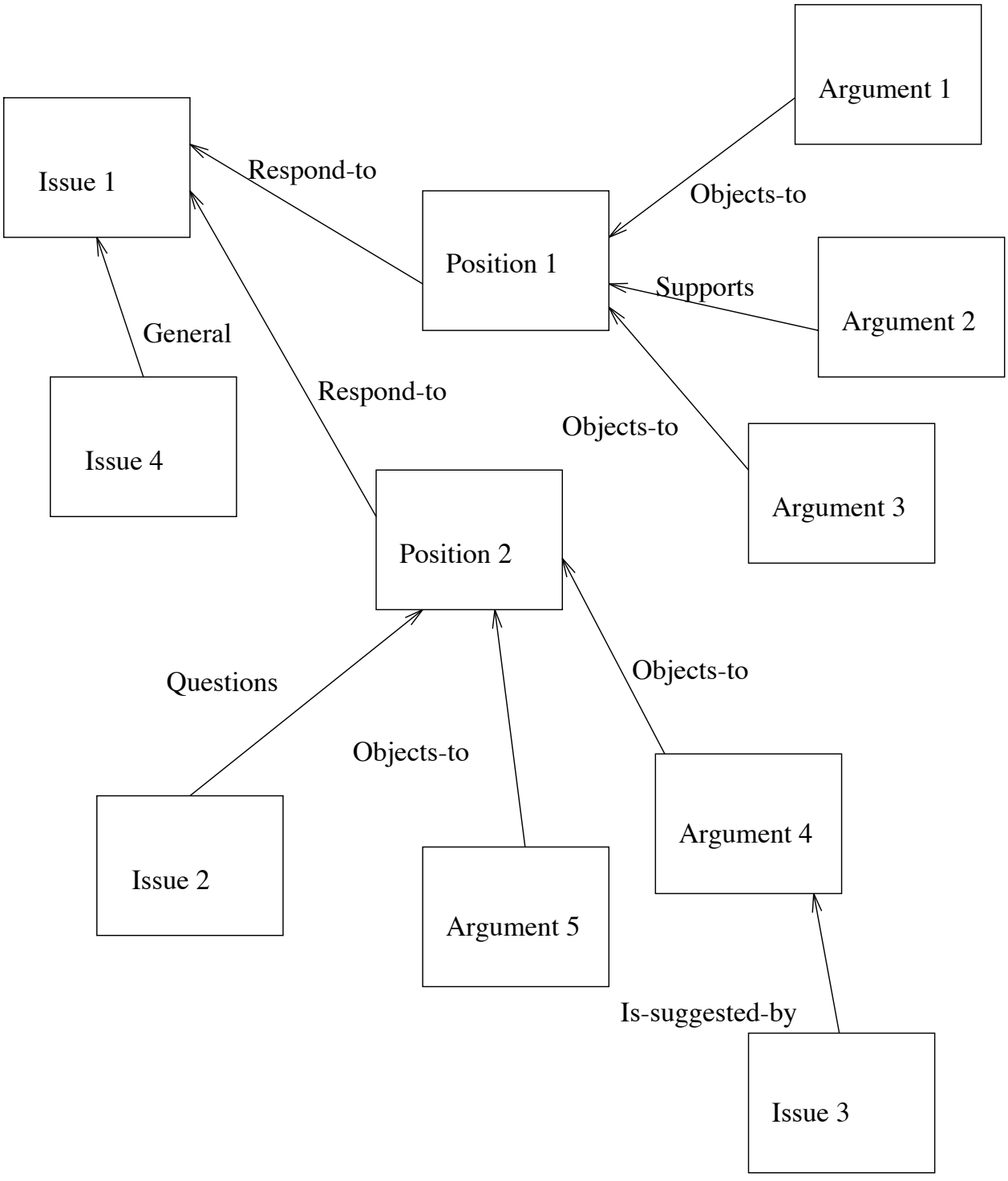


Figure 11: A segment of possible IBIS-style discussion network

ציור 11: קטע מתוך רשת שיחה בנוסח IBIS

In wicked problem exploration, understanding other viewpoints better allows understanding the whole problem better. IBIS supports design planning conversation. In IBIS, a basic device is stressed, a hypertext which enables many participants to navigate among the issues. There is no stopping rule, nor is there in the IBIS method a particular way of registering that an issue has been resolved by agreement upon some position. Rather, the goal of the discussion is for each of the stakeholders to try to understand the specific elements of each others' proposals.

The method makes it harder for discussants to make unconstructive rhetorical moves, such as "argument by repetition", and it supports other more constructive moves, such as seeking the central issue, asking questions as much as giving answers, and being specific about the support evidence of one's viewpoint.

Users of gIBIS [CB88], a hypertext tool for exploratory discussion that implements IBIS graphically, reported that the Issue-Position-Argument framework helped to focus their their thinking on the hard, critical parts of the problem and to detect incompleteness and inconsistency in their thinking more readily. Users reported that the structure that is imposed on discussions was very useful and served to expose "axe grinding, hand waving, and clever rhetoric". They also valued the tendency for assumptions and definitions to be made explicit. These features of a cooperative environment are very important for requirements elicitation.

It is usually the case that any change in a project impacts the software. So, whenever a change is suggested, it is necessary to see the effects of the change. Moreover, after several changes are made, there are times in which designers may want to reconstruct their arguments and positions through the development. The IBIS hypertext lays it all out there to make it less likely that an important issue position or argument will be overlooked. The IBIS method is good for managing changes in requirements.

To sum up, the elicitor

- identifies conflicts or absence of information by examining the data inside the abstraction nodes,
- navigates associatively between abstractions nodes by using links,
- verifies information from different abstractions by using nodes and links,
- negotiates by using nodes and links, and
- updates the abstraction network by addition and removal of nodes and links.

Thus, the main issues in generating an abstraction network are

1. to segment the information into abstractions with the help of AbstFinder, and
2. to determine the types of links needed for decisions, updates, etc. as suggested above.

5.2 Problem Exploring with Hypertext

The concept of hypertext is quite simple. Windows on the screen are associated with objects in a data-base, and links are provided between these objects, both graphically, as labeled tokens, and in the database, as is shown in Figure 12.

The intent is to organize the initial abstractions on a hypertext, in order to give the elicitor the ability to author the huge and complex specifications. Hypertext gives the elicitor the ability to navigate through the information in any associative non-linear way he or she wants. However, there is the danger of creating such a complex hypertext that the elicitor gets lost in hyperspace.

5.2.1 Hypertext-Related Work

Hypertext, in general, is a research area of its own [Con87], This research area includes questions of navigation techniques, links carrying functions, and updating the hypertext (nodes and links). The general problem faced by all users of hypertext, is the question of getting lost in hyperspace. These questions are outside the scope of this exploration, since it is being dealt with as part of other more direct hypertext research [YMvD85, Con87, CB88].

The unresolved problem in hypertext is how to generate the hypertext in the first place. How to segment the data in oder to be able to use efficiently the associative navigation feature of the hypertext. Usually, the user is responsible for doing this segmentation. gIBIS [CB88] is a hypertext tool for exploratory discussion that implements IBIS graphically, as mentioned before. Note, that Conklin says there that they are still considering the addition of a brainstorming mode, in which they will try to overcome the cognitive overhead required to *segment* the "mass of information" (muck, in Conklin's words) into discrete thoughts, identify their types, label them, and link them. AbstFinder is a tool that assists in segmenting the information according to abstractions.

Recently, a great deal of work in software engineering is taking advantage of hypertext [DS87, GS87]. The problem addressed in this work is maintaining software life cycle on a hypertext, keeping links between code and its documentation and links between the requirements to different products of software life cycle. Indeed, Garg and Scacchi have suggested maintaining all life-cycle documents as hypertext [GS87, Gar89].

There are some tools which are more modelling oriented. READS [Smi93] is an hypertext system designed to support the key requirements engineering activities of requirements discovery, analysis, decomposition, allocation, traceability and reporting. READS facilitates the construction, browsing, and maintenance of a typed hypertext network with a user interface designed specifically for the system engineer. But, as quoted from the paper, "Requirement discovery and extraction are done by examination of the document through scrolling and regular expression searching. Candidate requirement statements are selected with the mouse and placed into the requirements inspection window from which they may be saved into the project database." Still, the process of capturing the requirements in the first place is done by humans.

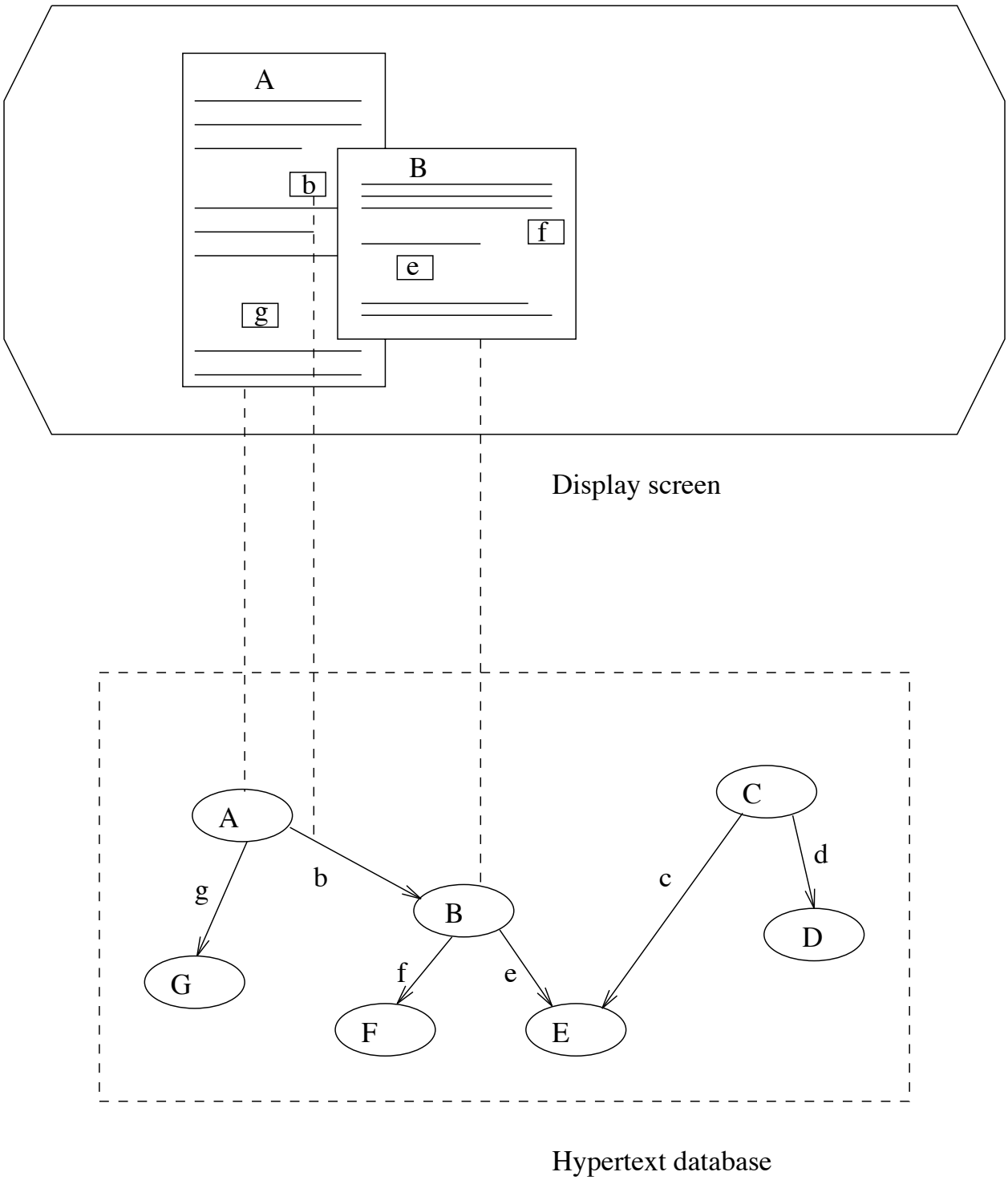


Figure 12: The hypertext concept
ציור 12: מושג ה-hypertext

AbstFinder can clearly complement READS.

RETH, Requirements Engineering Through Hypertext (See Section 1.4) [Kai93], also uses a hypertext that helps the users and analysts to make relationships and dependencies explicit. RETH helps to gather and structure the requirements only by supporting *brainstorming* through hypertext. For capturing the raw requirements, Kaindl suggests "to get help from an analyst (a requirements engineer)" since requirements formation is "too difficult for inexperienced users".

5.2.2 Generate Abstraction Network On Top of Hypertext

In order to build the abstraction network on the top of some hypertext, it is needed to transform the initial interview documents from each user into a hypertext organized in a way that represents whatever relations the elicitor desires.

There are now two kinds of links and maybe these are all that are ever needed.

a. One kind are structural links that cannot be generated automatically. These capture that an abstraction is a subclass of another, that one is dependent on the other, that one is a component of the other, or maybe even all of those. For example, this structure can be that of the decomposition, as was done for findphrases by Aguilera for implementation purposes. Such a structure cannot be generated automatically.

b. The other kind are semantic links, which link together all sentences that talk about same abstraction, or which link the use of a word and its defining module. These can be generated automatically as described before.

It is suggested to show only the structural links graphically because they are not automatically generated and need human thinking to make. Therefore, showing them gives more information. Showing automatically generated links does not give new information.

The following is a list of needed operations, both automatic and manual by the elicitor, that help to prepare the abstraction network on top of hypertext for the elicitation process.

1. segment data into abstraction nodes (manually assisted by invoking UNIX scripts on the AbstFinder's output list of abstractions identifiers),

2. generate cross reference type links (automatically from the information generated by AbstFinder),

3. import nodes to hypertext (automatically by invoking UNIX scripts that generate the markup language of a hypertext),

4. generate structural type links (manually by the elicitor)

5. navigation (manually by highlighted words, buttons, log and history-hypertext features)

6. update links and nodes (out of the scope of this thesis, as it is the subject of other hypertext research)

Step 1, 2, and 3 will be demonstrated in the following case study.

5.2.3 An Abstraction Network Case Study

An abstraction-oriented network was prototyped on the top of Hyperties, a hypertext system, using findphrases as a case study. The purpose of the case study is to demonstrate the initial generation of an abstraction network, with the assistance of AbstFinder and the hypertext features, and to explore whether the assumptions about desirable operations are valid.

Using Hyperties, an on-the-shelf system allows focusing on the abstraction organization problem rather than getting bogged down in the details of building a hypertext system. Hyperties is a powerful software tool for organizing and presenting information. It was developed at the University of Maryland's Human computer Interaction Laboratory [KS88, HYP91].

Hyperties consists of two programs, the Authoring System and the Browser. The Authoring System is used to create a database of articles and illustrations. Using the authoring system is like using a familiar word processor. The author types in the text of the articles and specifies the links or cross references to other articles and illustrations. Hyperties automatically ties the articles and illustrations together into a unified database and constructs an index to the entire database.

The Browser enables readers to access a Hyperties database of articles and illustrations. Using the browser is extremely easy and requires virtually no training. The reader can follow a topic of interest, turn pages (*Next* or *Backup*) (See Appendix F Figure 15-d). The power of Hyperties comes from the links that tie articles and illustrations together. A link is a cross reference, an indication that more information articles and illustrations. Hyperties automatically keeps track of the path followed by the browsing reader so that he or she can return to previously seen articles.

Hyperties automatically creates an index which lists all the articles in the database. Readers may go to the index at any time and access any article in the database directly.

Authors may wish to refer to the same article using different phrases as links. The author need not plan this in advance; as Hyperties builds its index it will ask if certain terms are to be considered synonyms or not. Hyperties enables the user to VIEW the output through the BROWSER. This feature is very convenient for the author who wants to see how the output should or would look. The change cycle is very short and convenient.

The hypertext that was used for the experiment was Hyperties 3.0 standard version that comes with a set of pre-defined visual designs. The user cannot add his or her own designs. There is a Hyperties 3.0 professional version that provides the author the ability to create his or her own designs.

The IMPORT utility is a program to facilitate bringing documents from other sources into a Hyperties database. Its ability to bring existing files into a database in one step, and resolve links as it does so, makes it a very powerful tool. The documents to be imported using this utility must include mark-up commands to tell IMPORT how to handle the article in the Hyperties database. A document including mark-up commands is referred to as an AIM (Automatic IMport) file.

The following describes the three initial steps in the process of restructuring knowledge for abstraction oriented network, used for requirements elicitation.

1. Generate an article for each abstraction node in the hypertext. The abstraction content is the set of all the sentences from the original document that contain words of the abstraction identifier as recognized by AbstFinder. This is done by:

   ```
   agrep corr-phrases-of abstraction original-transcript > abstraction-article
   ```

2. Any part of some abstraction identifier, in the form of complete words, that is contained in the content of some other abstraction is a candidate for being a name of a cross reference type link.

3. Create a new database with Hyperties, and IMPORT the marked-up articles into it.

5.2.3.1 Findphrases Case Study

The findphrases was already used as a case study for AbstFinder, and the the full abstraction list of findphrases, approved by the designer of findphrases, Aguilera is available. Each abstraction, as an output of AbstFinder, is identified by a phrase, which is the correlated phrase that caused the similarity between the sentences that compose the abstraction. So, the nodes of the abstraction network are well identified.

The next sections describe the the initial building of the abstraction network of the findphrases case study, as described previously.

1. The abstractions are well identified. Thus, it remains to collect all the sentences that belong to an abstraction into an article that will be the content of the abstraction node. This is done by agrep, that collects the sentences, one per line, that match the identifying phrase from the input transcript into a file called article in the following. Now, article is composed of edited sentences extracted from the transcripts to which AbstFinder was applied.

2. Those correlated phrases that identify the abstractions, actually define the cross reference abstractions links that should be generated automatically. In each abstraction, where one or more of those phrases (that identify any abstraction) appears, a link should be built automatically, between the abstraction that contains the phrase to the abstraction that is identified by that phrase. Recall that a result of a run of AbstFinder is the *corr-phrases-file*, the file that contains the correlated phrases that identify all the abstractions. For each

abstraction *i*, a copy was kept of all correlated phrases that identify all the abstractions found minus the correlated phrases that identify *i* itself, i.e., the phrases that identify all other abstractions. Then, in each abstraction article, the mark-up that would make those phrases links was generated. This mark-up is done by surrounding the phrase link with <L> and <\L>, using the make utility (See Appendix F.1).

3. Now, the marked-up abstraction article is ready to be IMPORTED into the Hyperties database.

At this point, the basic authoring of the abstraction network was done, ready for the elicitor to browse the abstraction database. It seems that an hypertext is the way to go at least for smaller problems. This idea was explored by the findphrases case study. Note that the purpose of this case study is to test the functionality of the abstraction network and not for value conclusions. Appendix F contains highlights and snapshots of browsing in the abstraction network of the findphrases case study.

5.3 Abstraction Network Exploration Summary

It seems that an hypertext is a good basis for an abstraction network. As originally envisioned, the individual nodes would be pure text. It seems that for larger problems, more structure will be needed in the interiors of nodes, which will be populated also by structural links. For instance, in the RFP case study, there is an abstraction "testing". Zooming allowed finding its sub-abstractions. In such a case, instead of the long pure text of the "testing" abstraction, it may be better to build separate sub-abstractions, e.g. "acceptance tests", "built-in test", "environment tests", etc., and link them with structural links to their parent "testing". This can easily done in Hyperties by building an abstraction which is actually an indexed list of links to the sub-abstractions.

A major issue was what kinds of links should be generated automatically and in response to what operations should these be generated. Only cross-reference type links can be generated automatically, as described in Section 5.2.3.1. Cross-reference type links are those that link all sentences that talk about the same abstraction or that link the use of a word with its its defining module.

Other questions to be asked in the future during real-life applications of the tools include the following.

1. Are there other operations that should cause the creation of links?

2. Should the links be made quietly?

3. Should the user be asked if he or she wants the links?

4. Should the links should be made only upon explicit request of the user?

5. Is hypertext even appropriate?

6. If not, then what is appropriate?

This research stops at the point in which the elicitor begins writing the insides of the abstractions, in order to write the requirements of the system which is the subject of the interviews. The research stops at this point because the problem is now one of analysis and no longer one of elicitation. Analysis has been thoroughly studied by others [RS77, SM85, TH77, WE82, Alf77, Zav82, BGM85].

6 Conclusions

The work reported in this dissertation was aimed at designing, prototyping, and determining the effectiveness of an essential part of an envisioned integrated environment, REGAE, for gathering, sifting, and writing requirements. REGAE is described as helping the human requirements elicitor massage transcripts of interviews with members of a client organization into a consistent, complete, unambiguous, coherent, and concise statement of what the organization wants. The goal of REGAE is to organize the whole collection of requirements information as a network, of nodes each denoting an abstraction and containing a description of all that is known and required about the abstraction.

To arrive at the basis for a tool that helps identify the abstractions that will make the nodes from the transcripts of the interviews, it was necessary to ask what are the most effective ways of identifying abstractions in natural languages transcripts of client interviews.

Preliminary studies described in Sections 2, 2.2.2, and 2.2.3 indicated that a repeated phrase finder and a lexical affinities finder are both effective for small problems. They each have weaknesses both that the other does not have and that the other does have. The work has considered a new tool, AbstFinder, which overcomes most of the former tools' weaknesses. A number of case studies demonstrated the effectiveness of AbstFinder even on large problems (See Section 4.2.3).

1. The findphrases case study (See Section 2.2.2) helped test the accuracy of the tool in a small, previously tested problem. The results of the application of the older tools mentioned above to this problem gave a means to determine the the effectiveness of the new tool in comparison to that of the older ones.

2. The RFP case study involved applying AbstFinder to a real-life, industrial strength, full sized problem. AbstFinder was applied to a variety of customer-generated requirement documents in order to test the coverage of the abstractions found, and their completeness.

The key measures of the effectiveness of AbstFinder are: (1) its coverage, and (2) how summarizing it is. A tool that is not covering or which does not summarize is not good. Heretofore, abstraction identification has been done manually by a human elicitor. The problem is that humans get tired, get bored, fall asleep, and overlook relevant ideas. Therefore, an elicitor will not be willing to be assisted by any tool unless he or she is confident that it is covering, that no critical piece of information has been overlooked in the process of abstraction identification. The case studies have shown AbstFinder to be covering (See Sections 2.2.2 and 4.2.3).

A tool that is only covering is not very helpful. The most covering tool is that which outputs everything that is input, and the elicitor is left right where he or she started. Therefore, the other main requirement for the tool is that it reduce the amount of text that the elicitor must look at. An elicitor still has to do the *thinking* with the output of the tool, in order to approve the abstractions found. The case studies have shown that AbstFinder is indeed summarizing. In the RFP case study, the output was about 21% of the size of the input data (See Section 4.2.3). Note

finally, that a tool that is only summarizing is no good either. The most summarizing tool is that which outputs nothing. The tool must summarize while preserving coverage, and AbstFinder is doing precisely that for the case studies.

Finally, in the exploration of the use of hypertext to build the abstraction network it was determined that only cross reference links should be built automatically, leaving other links to be discovered and built by the thinking elicitor.

Bibliography

[AB89] Abe, K.K. and Berry, D.M., "indx and findphrases, A System for Generating Indexes for Ditroff Documents," *Software—Practice and Experience* **19**(1), p.1–34 (1989).

[AB90] Aguilera, C. and Berry, D.M., "The Use of a Repeated Phrase Finder in Requirements Extraction," *Journal of Systems and Software* **13**(9), p.209–230 (1990).

[Abb83] Abbott, R.J., "Program Design by Informal English Descriptions," *CACM* **26**(11) (November, 1983).

[Agu87] Aguilera, C.S., "Finding Abstractions in Problem Descriptions using findphrases," M.S. Thesis, Computer Science Department, UCLA, Los Angeles, CA (October, 1987).

[Alf77] Alford, M.W., "A Requirements Engineering Methodology for Realtime Processing Requirements," *IEEE Transactions on Software Engineering* **SE-3**(1), p.60–69 (1977).

[Alf79] Alford, M.W., "Software Requirements Engineering Methodology (SREM) at the Age of Two," in *COMPSAC 78 Proceedings* (November, 1978).

[Alf85] Alford, M.W., "SREM at the Age of Eight; The Distributed Computing Design System," *Computer* **18**(4), p.36–46 (April, 1985).

[ALW91] Agrawala, A.K., Lavi, J.Z., and White, S.W., "Task Force on Computer-Based System Engineering Holds First Meeting," *IEEE Computer* **24**(8), p.86–87 (August, 1991).

[BGM85] Borgida, A., Greenspan, S., and Mylopolous, J., "Knowledge Representation as the Basis for Requirements Specifications," *Computer* **18**(4), p.82–91 (April, 1985).

[BGN86] Berzins, V., Gray, M., and Naumann, D., "Abstraction-Based Software Development," *Communications of the ACM* **29**(5), p.402–415 (May, 1986).

[Boe81] Boehm, B.W., *Software Engineering Economics,* Prentice-Hall, Englewood Cliffs, NJ (1981).

[Boe88] Boehm, B.W., "A Spiral Model of Software Development and Enhancement," *IEEE Computer* **21**(5), p.61–72 (May, 1988).

[Boo86] Booch, G., *Software Engineering with Ada,* Benjamin-Cummins, San Francisco, CA (1986). Second Edition.

[Bur84] Burstin, M.D., "Requirements Analysis of Large Software Systems," Ph.D. Dissertation, Department of Management, Tel Aviv University, Tel Aviv, Israel (1984).

[BYC90] Burgess-Yakemovic, K.C. and Conklin, J., "Report on Development Project Use of an Issue-Based Information System," in *Proceedings of the ACM Conference on CSCW*, Los Angeles, CA (October, 1990).

[BYY87] Berry, D.M., Yavne, N.M., and Yavne, M., "Application of Program Design Language Tools to Abbott's Method of Program Design by Informal Natural Language Descriptions," *Journal of Software and Systems* **7**, p.221–247 (1987).

[CASE88] *IEEE Software* **5**(2) (March, 1988).

[CB88] Conklin, J. and Begeman, M.L., "gIBIS:A Hypertext Tool for Exploratory Policy Discussion," *ACM Transactions on Office Information Systems* **6**(4), p.303–331 (October, 1988).

[Con87] Conklin, J., "A Survey of Hypertext," MCC Technical Report No. STP-356-86, Rev. 1, MCC, Austin, TX (February 9, 1987).

[Cru86] Cruse, D.A., *Lexical Semantics,* Cambridge University Press, Cambridge (1986).

[CSCW88] CSCW, *Proceedings of the Conference on Computer-Supported Cooperative Work*, September 26-29, 1988. Anonymous.

[CWS93] Christel, M.G., Wood, D.P., and Stevens, S.M., "AMORE - The Advanced Multimedia Organizer for Requirements Elicitation," Technical Report,CMU/SEI-93-TR-12, Software Engineering Institute (June, 1993).

[Dav90] Davis, A.M., *Software Requirements: Analysis and Specification,* Prentice-Hall, Englewood Cliffs, NJ (1990).

[DOD85] "Specification Practices," MIL-STD 490A, U.S. Department of Defense (June 4, 1985).

[DS87] Delisle, N.M. and Schwartz, M.D., "Contexts — A Partitioning Concept for Hypertext," *ACM Transactions on Office Information Systems* **5**(2), p.168–186 (April, 1987).

[Eas93] Easterbrook, S., "Domain Modelling with Hierarchies of Alternative Viewpoints," pp. 65–72 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[EFRV86] Estrin, G., Fenchel, R.S, Razouk, R.R., and Vernon, M.K., "SARA (System ARchitect's Apprentice): Modeling, Analysis, and Simulation Support for Design of Concurrent Systems," *IEEE Transactions on Software Engineering* **SE-12**(2), p.293–311 (1986).

[Fea93] Feather, M.S., "Requirements Reconnoitering at the Juncture of Domain and Instance," pp. 73–76 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[FKN92] Finkelstein, A., Kramer, J., and Nuseibeh, B., "Viewpoints: A framework for Integrating Multiple Perspectives in System Development," *International Journal of Software Engineering and Knowledge Engineering* **2**(1), p.31–57 (1992).

[Gar89] Garg, P.K., "GRAP—Information Management in Software Engineering: A Hypertext Based Approach," Ph.D. Dissertation, Computer Science Department, University of Southern California, Los Angeles, CA (February, 1989).

[GL93] Goguen, J.A. and Linde, C., "Techniques for Requirements Elicitation," pp. 152–164 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[GS87] Garg, P.K. and Scacchi, W., "Maintaining Software Life Cycle Documents as Hypertext: Issues, Analysis, and Directions," Technical Report, University of Southern California, Los Angeles, California (1987).

[Har87] Harel, D., "On Visual Formalisms," *Communications of the ACM* **30**(6) (June, 1987).

[HJ90] Holtzblatt, K. and Jones, S., "Contextual Inquiry: Principles and Practice," Technical Report DEC-TR 729, Digital Equipment Corporation (October, 1990).

[HLNPPSST90] Harel, D., Lachover, H., Naamad, A., Pnueli, A., Politi, M., Sherman, R., Shtul-Trauring, A., and Trakhtenbrot, M., "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *IEEE Transactions on Software Engineering* **SE-16**, p.403–414 (1990).

[Hud84] Huddleston, R., *Introduction to the Grammar of English,* Cambridge University Press, Cambridge (1984).

[HYP91] *HYPERTIES Version 3.0, User's Guide,* Cognetics Corporation (1987-1991).

[IAI89] "System Specification for Unmanned Air Vehicle — Short-Range (UAV-SR) System (RFP)," Technical Report, Israeli Aircraft Industries, Ltd. (1989).

[IAI90] "Allocated Software System Requirements for Unmanned Air Vehicle — Short-Range (UAV-SR) System (ASSR)," Technical Report, Israeli Aircraft Industries, Ltd. (1990).

[IEEE92] IEEE, "IEEE Standard Glossary of Software Engineering Terminology," in *ANSI/Standard 610.12*, IEEE, New York, NY (1992). Anonymous.

[ISK93] Ishihara, Y., Seki, H., and Kasami, T., "A Translation Method from Natural Language Specifications into Formal Specifications Using Contextual Dependencies," pp. 232–239 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[Jac75] Jackson, M.A., *Principles of Program Design,* Academic Press, London (1975).

[Kai93] Kaindl, H., "The Missing Link in Requirements Engineering," *ACM SIGSOFT Software Engineering Notes* **18**(2), p.30–39 (April, 1993).

[KCHNP90] Kang, K., Cohen, S., Hess, J., Novak, W., and Peterson, A., "Feature Oriented Domain Analysis (FODA) Feasibility Study," Technical Report, CMU/SEI-90-TR-21, DTIC: ADA235785, Software Engineering Institute (November, 1990).

[Knu73] Knuth, D.E., *The Art of Computer Programming: Sorting and Searching,* Addison-Wesley, Reading, MA (1973).

[Kra88] Krasner, H., "Requirements Problems in Large Software Projects: New Directions for Software Engineering Technology," Technical Report, MCC, Austin, TX (1988).

[KS88] Kreitzberg, C.B. and Shneiderman, B., "Restructuring Knowledge for an Electronic Encyclopedia," in *International Ergonomics Association Tenth Congress*, Sydney, Australia (1988).

[Lehman 1980] Lehman, M.M., "Programs, Life Cycles, and Laws of Software Evolution," *Proceedings of the IEEE* **68**(9), p.1060–1076 (September, 1980).

[Lei87] Leite, J.C.S.P., "A Survey on Requirements Analysis.," Advanced Software Engineering Project Technical Report RPT-071, Department of Information and Computer Science, University of California, Irvine, CA (June, 1987).

[LF91] Leite, J.C.S.P. and Freeman, P., "Requirements Validation through Viewpoint Resolution," *IEEE Transactions on Software Engineering* **SE-17**(12) (December, 1991).

[LF93] Leite, J.C.S.P. and Franco, A.P.M., "A Strategy for Conceptual Model Acquisition," pp. 243–246 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[LPR93] Lubars, M., Potts, C., and Richter, C., "A Review of the State of Practice in Requirements Modeling," pp. 2–14 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[Luh58] Luhn, M., "The Automatic Creation of Literature Abstracts," *IBM Journal of Research and Development* **2**(2), p.159–165 (April, 1958).

[LW89] Lavi, J.Z. and Winokur, M., "ECSAM - A Method for the Analysis of Complex Computer Systems & their Software," in *Proceedings of the Fifth Structured Techniques Association Conference*, Chicago (May, 1989).

[Maa89] Maarek, Y., "Using Structural Information for Managing Very Large Software Systems," Ph.D. Dissertation, Faculty of Computer Science Department, Technion, Haifa, Israel (January, 1989).

[MB88] Maarek, Y. and Berry, D.M., "The Use of Lexical Affinities in Requirements Extraction," Technical Report, Technion, Haifa, Israel (November, 1988).

[McD89] McDermid, J.A., "Requirements Analysis: Problems and the STARTS Approach," pp. 4/1–4/4 in *IEEE Colloquium on Requirements Capture and Specification for Critical Systems (Digest No. 138)* (November, 1989).

[Mit84] Mitchell, W., *A Prelude to Programming: Problem Solving and Algorithms,* Reston Publishing, Reston, VA (1984).

[MT88] Martin, J. and Tsai, W.T., "An Experimental Study in Upstream Software Development," Technical Report, University of Minnesota, Minneapolis, MN (1988).

[Mye79] Myers, G.J., *Composite/Structured Design,* van Nostrand Reinhold, New York, NY (1979).

[NR69] "Software Engineering: Report on a Conference Sponsored by the NATO Science Commission," Garmisch, Germany, 7-11 October 1968, Scientific Affairs Division, NATO, Brussels, Belgium (January, 1969).

[Orr77] Orr, K.T., *Structured Systems Development,* Yourdon, New York (1977).

[Orr81] Orr, K.T., *Structured Requirements Engineering,* Ken Orr & Associates, Topeka, KS (1981).

[Parnas 1972] Parnas, D.L., "On the Criteria to be Used in Decomposing Systems into Modules," *Communications of the ACM* **15**(2), p.1053–1058 (December, 1972).

[PCCW93] Paulk, M.C., Curtis, B., Chrissis, M.B., and Weber, C.V., "Key Practices of the Capability Maturity Model," Technical Report, CMU/SEI-93-TR-25, Software Engineering Institute (February, 1993).

[PT93] Potts, C. and Takahashi, K., "An Active Hypertext Model for System Requirements," Technical Report, College of Computing, Georgia Institute of Technology (1993).

[RE93] S. Fickas and A. Finkelstein, Eds., *Proceedings of the IEEE International Symposium on Requirements Engineering,* IEEE Computer Society, San Diego, CA (January 4-6, 1993).

[Ros77] Ross, D.T., "Structured Analysis (SA): A Language for Communicating Ideas," *IEEE Transactions on Software Engineering* **SE-3**(1), p.16–33 (January, 1977).

[RS77] Ross, D.T. and Schoman, K.E. Jr., "Structured Analysis for Requirements Definition," *IEEE Transactions on Software Engineering* **SE-3**(1), p.6–15 (January, 1977).

[Rya93] Ryan, K., "The Role of Natural Language in Requirements Engineering," pp. 240–242 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[Sal89] Salton, G., *Automatic Text Processing: The Transformation, Analysis and Retrieval of Information by Computer,* Addison-Wesley, Reading, MA (1989).

[Sch92] Schach, S.R., *Software Engineering,* Aksen Associates & Irwin, Boston, MA (1992). Second Edition.

[SEG68] Sackman, H., Erickson, W.J., and Grant, E.E., "Exploratory Experimental Studies Comparing Online and Offline Programming Performance," *Communications of the ACM* **11**(1), p.3–11 (January, 1968).

[SHTUE87] Saeki, M., Horai, H., Toyama, K., Uematsu, N., and Enomoto, H., "Specification Framework Based on Natural Language," pp. 87–94 in *Proceedings of the Fourth International Workshop on Software Specification and Design*, Monterey, CA (April, 1987).

[SK83] Sankoff, D. and Kruskal, J.B., *Time Warps, String Edits, and Macromolecules: The Theory and Practice of sequence Comparison,* Addison-Wesley, Reading, MA (1983).

[Skl88] Sklar, B., *Digital Communication Fundementals and Applications,* Prentice-Hall, Englewood Cliffs, NJ (1988).

[SM83] Salton, G. and McGill, M.J., *Introduction to Modern Information Retrieval,* McGraw-Hill, New York (1983).

[SM85] Sievert, G.E. and Mizell, T.A., "Specification-Based Software Engineering with TAGS," *Computer* **18**(4), p.56–66 (April, 1985).

[Smi93] Smith, T.J., "READS: A Requirements Engineering Tool," pp. 94–97 in *Proceedings of the IEEE International Symposium on Requirements Engineering*, San Diego, CA (January, 1993).

[SMT92] Schneider, G.M., Martin, J., and Tsai, W.T., "An Experimental Study of Fault Detection in User Requirements Documents," *ACM Transactions on Software Engineering and Methodology* **1**(2), p.188–204 (April, 1992).

[SSKLW90] Siegel, J.A.L., Stewman, S., Konda, S., Larkey, P.D., and Wagner, W.G., "National Software Capacity: Near-Term Study," Technical Report, CMU/SEI-90-TR-12, Software Engineering Institute (1990).

[SSR85] Scheffer, P.A., Stone, III, A.H., and Rzepka, W.E., "A Case Study of SREM," *Computer* **18**(4), p.47–54 (April, 1985).

[TH77] Teichroew, D. and Hershey, E.A. III, "PSL/PSA: A Computer-Aided Technique for Structure Documentation and Analysis of Information Processing Systems," *IEEE Transactions on Software Engineering* **SE-3**(1), p.41–48 (January, 1977).

[WAHKMOOTW93] White, S., Alford, M., Hotlzman, J., Kuehl, S., McCay, B., Oliver, D., Owens, D., Tully, C., and Willey, A., "Systems Engineering of Computer-Based Systems," *IEEE Computer* **26**(11), p.54–65 (November, 1993).

[WE82] Winchester, J. and Estrin, G., "Requirements Definition and Its Interface to the SARA Design Methodology for Computer-Based Systems," *AFIPS Conference Proceedings* **51**, p.369–379 (June, 1982).

[WLLO90] Winokur, M., Lavi, J.Z., Lavi, I., and Oz, R., "Requirements Analysis and Specifications of Embedded Computer Systems using ECSAM - a Case Study," pp. 80–89 in *Proceedings of the IEEE CompEuro Conference*, Tel Aviv, Israel (May, 1990).

[WM91] Wu, S. and Manber, U., "Fast Text Searching With Errors," TR 91-11, Computer Science Department, University of Arizona, Tucson, AZ (June, 1991).

[WS84] Wiener, R. and Sincovec, R., *Software Engineering with Modula-2 and Ada,* John Wiley & Sons, New York (1984).

[YMvD85] Yankelovich, N., Meyerowitz, N., and Dam, A. van, "Reading and Writing the Electronic Book," *Computer* **10**(18), p.15–30 (October, 1985).

[Zav82] Zave, P., "An Operational Approach to Requirements Specification for Embedded Systems," *IEEE Transactions on Software Engineering* **SE-8**(3) (May, 1982).

Appendix A AbstFinder Implementation Considerations

A.1 AbstFinder Output

If AbstFinder identifies, for example, "|surface |metal |" as an abstraction, it means that both phrases appeared in the sentence that served as the reference for comparison to all other sentences. "|surface |metal |", appearing in the AbstFinder output, means that the abstraction contains sentences that contain only "surface", sentences that contain only "metal", and sentences that contain them both, but as different runs. This means that "metal surface", as interpreted by the elicitor, will serve as the identifier of the abstraction, and the content of the abstraction will be the union of sentences that contain "metal" and sentences that contain "surface".

corr-phrases-file is the output file of AbstFinder that contains the pure list of the abstraction identifiers, i.e., the "correlated-phrases" that appear in the right most column of AbstFinder formatted output. Each line *i* on *corr-phrases-file* contains the correlated-phrases that identify abstraction *i*. These identifiers are phrases in the free language consisting of the original transcript words of fragments thereof. The elicitor will scan those identifiers in order to approve the abstractions. All the other information in the formatted output is for research purposes only.

Repetitions and Estimation in the Output

By explicit design decisions there are duplicates in the output. The design decisions derive from the requirement that the tool be clerical and not attempt any intelligence that might lose coverage. Two kinds of repetitions are identified:

1. repetition of a term within an abstraction's identifier, such as, "s file|tokens file|" (See Appendix C.3). Suppose the reference sentence, the one which is compared to the others, contains the phrase "xxx tokens file xxx". In the rest of the document there are sentences that contain only "file", and there are sentences that contain the full phrase "xxx tokens file xxxx". So, it is possible that the phrases that caused similarity will be found in full, as in the reference sentences, and also in a substring. Note that sometimes part of the correlated-phrases will be one character noise, e.g., "...s ...". This noise happens because AbstFinder ignores word boundaries. The elicitor should ignore the noise. There is also the possibility that in one sentence appears a concept and in two or more phrases, and the elicitor will interpret the term as only one phrase.

2. repetitions of abstraction identifiers, such as "equipment" (See Appendix E.3) Sometimes the same abstraction "equipment" appear in six different lines. Suppose, for example, "equipment" appears in sentences 3, 5, and 8. Then, when 5 is used as the reference, the abstraction (5,3,8) is also identified by "equipment". Then, in the same way, when 8 is used as the reference, (8,3,5) is found also identified by "equipment". These are actually permutations of the same abstraction. Since coverage was the most important criteria in this research, no attempt was done to clean repetitions from the output list of AbstFinder. Now that the prototype has successfully proved a concept, it is time to consider

adding filters for cleaning repetitions from the output, but the filters have to be designed carefully to insure that coverage is not lost.

The prototype was built in order to check the feasibility of a new method. The elicitor was counted on doing *all* the thinking and overcoming reasonable inconveniences. In the future, filtering utilities can be implemented for cleaning repetitions in the output.

An explanation is in order about the value of "corr_lines#", e.g., the number of sentences (See also Appendix A.3) contained in an abstraction. If some abstraction is identified by four phrases, for example abstraction number 426 in the RFP results (See Appendix E.3), "`| flight| flight |mination | flight termination |`". Then "corr_lines#" will count all lines that contain all those combinations, i.e., lines that have "`flight`" and "`mination`" and "` flight termination `" (blank before plus blank after). Thus, "corr_lines#" is greater than the number of occurrences of any of its individual pieces in the input.

The index "corr_lines#" is used for arranging the output, so that abstractions identified by fewer phrases will be at the top, and among the abstractions that have same "corr_phrase#", the ones with more "corr_lines#" will be higher in the output list. So, "corr_lines#" gives a rough estimate of how many times that abstraction appears in the transcript. Note again, "corr_lines#" is in the output for research purposes only. They are not to be used by the elicitor. The elicitor checks the whole list plus iterations in order to achieve full coverage. Moreover, when building the abstraction network, the elicitor chooses sentences according to one specific phrase. Then, the sentences can be counted accurately.

A.2 How does the Elicitor use AbstFinder Output

Confirm the Abstractions

The elicitor reads the AbstFinder output in order to capture the titles, abstraction identifiers, of the main ideas. Then, he or she goes back to the text in order to get from the original sentences in the original transcript all the details about that abstraction. Recall that AbstFinder outputs in part 2, the line numbers that are contained in each abstraction. The elicitor can use this list of line numbers, to go back to the original sentences for the details of the abstraction. However, this action is premature, since the abstraction network, is going to collect and display that information via a user-friendly interface. Moreover, these sentence numbers contain extra and irrelevant sentences, such as, for "`s file|token file|`", contain sentence with "`s file`" in addition to the relevant ones with "`token file`". Thus, the elicitor is suggested to concentrate on the abstractions identifiers, and to confirm that "token file" is the specific abstraction identifier in that case.

Interpretation of the Abstraction Title

If AbstFinder finds in the same shift two runs "a" and "b". The output will show them as "|a|b|" and not "|a b|". Since it is not known in what circular shift it will be found, than maybe the combination "b a" is the correct concept and not "a b". So, putting a bar between them reminds the reader to think of the right combination. AbstFinder outputs as one sequence only a complete run, which has in it the words in the order in the original transcript.

Not Useful Abstractions

In the design of the formatted output of AbstFinder, it was decided to indicate the point beyond which the abstractions are not useful in the following way. If the length of the list of "correlated-phrases" of some abstraction exceeds the page width, then it is truncated at the page boundary. The truncation is a visual clue to the elicitor that this abstraction is. The full information itself remains in part 2 of the AbstFinder formatted output just in case the presumption of uselessness is wrong.

A.3 AbstFinder Input Filtering

The original transcript is the text as written by the client in some word processor, with its own punctuation keywords. So, a sentence may go beyond a line or several lines. Because the AbstFinder algorithm is sentence oriented, the first step in AbstFinder program is to rearrange the input source into a sentence In the AbstFinder program the identification of sentence boundaries in the input transcript is done according to a user-input punctuation-keyword file. For instance, in a natural language transcript people usually use . ! ; or ? to define sentence boundaries. In many formal programming languages, the sentence delimiter will be ";".

Still there is a problem that people use a "." also for other purposes than punctuation. For instance, in section numbering, e.g., "1.2.11", and for abbreviations, e.g., "e.g.", "i.e.", "U.S.A.", "etc.". Since AbstFinder program uses standard filters for rearranging the input transcript, it will cut the section number "1.2.11" into three lines

1
2
11

and "e.g." will be cut to two lines

e
g

which will cause redundancy of line-sentences, and also may cut one sentence to several lines. Thus, an additional pre-pass was necessary on the input, in order to change the "." in those abbreviations or section numbering to some another keyword that is not a sentence delimiter in that transcript. The patterns of these changes are standard enough to make the the construction and use of this pre-pass part of the standard method of using AbstFinder. See Figure 13 for an overview of how to use AbstFinder, and see also Appendix B.

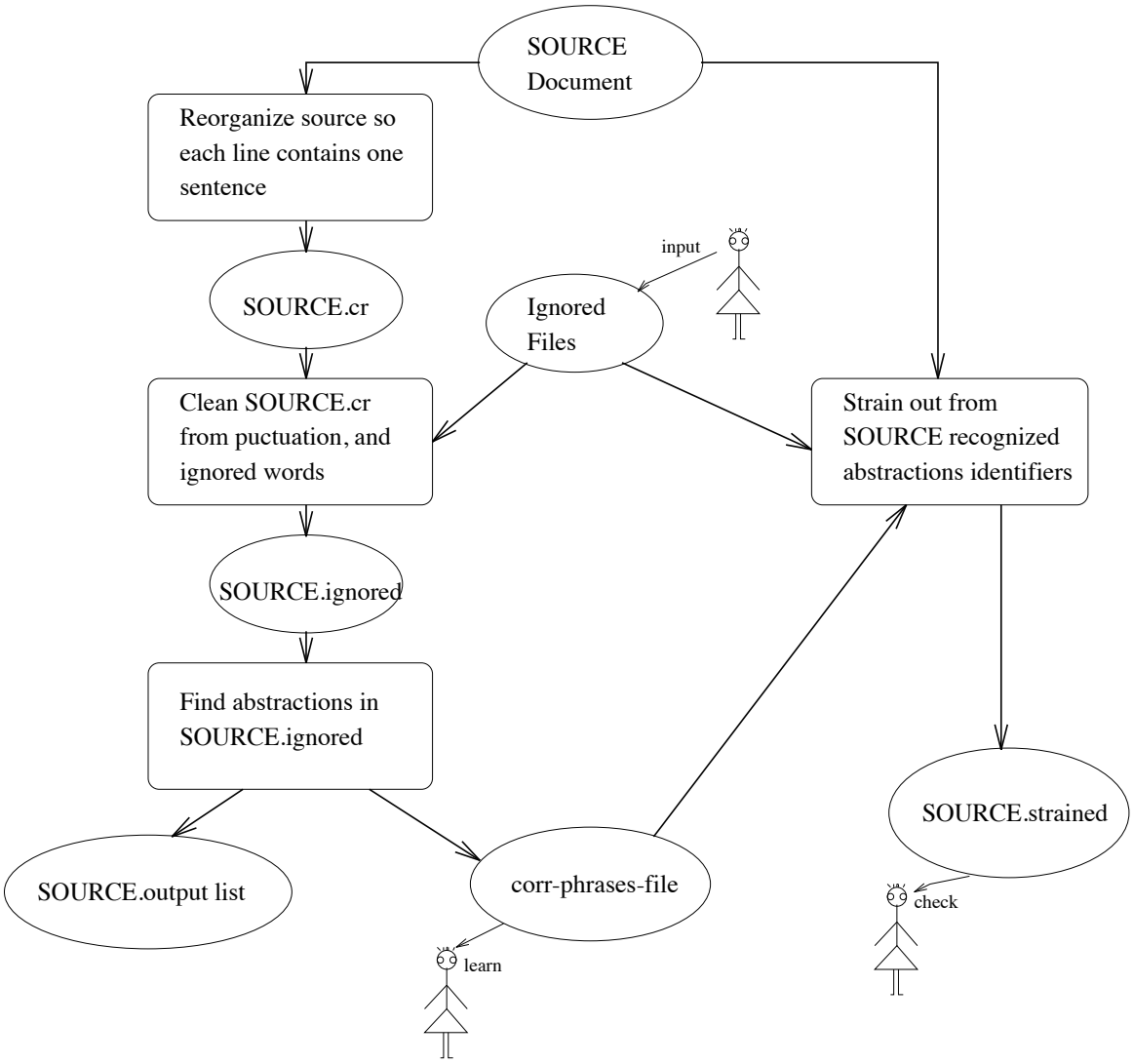


Figure 13: AbstFinder Input/Output Architecture
AbstFinder ציור 13: מבנה קלט פלט של

Appendix B Manual Pages of AbstFinder Tools

The manual pages for the AbstFinder Tools are in UNIX manual page format on the subsequent pages. Their page numbers have been edited to match their position in this thesis.

NAME

AbstFinder — A set of tools for abstraction identification

SYNOPSES

makeSentencesToLines *infile outfile*
eliminateIgnoredParts *infile outfile*
abstfinderFull *infile outfile* [*optional_new_name_for_CorrPhrases*]
abstfinderSummary *infile outfile* [*optional_new_name_for_CorrPhrases*]
strainTermsFromInput *infile outfile*

DESCRIPTION

AbstFinder is a collection of tools and an environment for producing a list of identified abstractions from an ASCII file written in any natural language or any formal language (programming or other). *AbstFinder* is based on the assumption that important abstractions are discussed more than once among the sentences written in the file. The more sentences contained in an abstraction the more significant it is.

First, the user invokes a shellscript, *makeSentencesToLines*, to reorganize the input file into lines, each containing one sentence according to a pre-defined set of punctuation characters.

Then, the user invokes a shellscript, *eliminateIgnoredParts*, to filter out from the input file ignored words taken from *IgnoredFile*, *IgnoredApplication*, and *IgnoredSuffixes*.

Finally, the user invokes the *abstfinderSummary* or *abstfinderFull* shellscript to identify the abstractions. The output of *abstfinderFull* is in two parts. The first part is a summary of the identified abstractions, and the second part contains a full description of the abstractions. *abstfinderSummary* gives on the first part and is usually sufficient for most purposes.

These steps may be repeated after updating *IgnoredApplication* with words discovered to skew the first output. Additionally, *strainTermsFromInput* may be run to verify that the bottom of the output list of terms that seems not to contain any new abstraction in fact does not.

ENVIRONMENT

The environment in which *AbstFinder* runs includes the following files:

IgnoredFile

IgnoredFile contains so-called noise phrases such as ‘‘`a`’’, ‘‘`an`’’, ‘‘`the`’’, ‘‘`of`’’, ‘‘`of the`’’, etc. plus any useless general phrases found in previous runs of the program. This file is intended to be used for any application.

IgnoredApplication

IgnoredApplication contains those frequent application keywords, which identify the abtraction of the application as a whole, such as the name of the application. These are to be ignored because their predominating presence overshadows other terms that are sought.

IgnoredSuffixes

IgnoredSuffixes contains suffixes such as ‘‘`tion`’’, ‘‘`ance`’’, ‘‘`ment`’’, ‘‘`ing`’’, ‘‘`able`’’, etc. These suffixes are marked by the program, *input*, so they are taken into consideration during the process of abstraction identification.

CorrPhrases

CorrPhrases is an output file containing those frequent phrases that were identified by *abstfinderSummary* or *abstfinderFull*. This file is used as an input file to the *strainTermsFromInput* shellscript.

USE OF TOOLS

Below is a description of a typical scenario for use of the tools; the file names beginning with “*source*” are particular for the example. Other file names are the same for all runs of the programs:

1. Prepare an input file for *AbstFinder*, say *source.in*

2. Run *makeSentencesToLines* on the input made in Step 1.

```
makeSentencesToLines source.in source.cr.
```

This command puts each sentence into its own line in *source.cr*.

3. In order to filter ignored words and ignored suffixes use *eliminateIgnoredParts* as follows:

a. Prepare *IgnoredFile* with common words to ignore, separated by blanks.

b. Prepare *IgnoredApplication* with application words to ignore, separated by blanks. This will usually be empty for the first run with a new application

c. Prepare *IgnoredSuffix* file with suffixes to ignore, separated by blanks.

d. To get the filtered input into a file, say *source.input*, do

```
eliminateIgnoredParts source.cr source.input.
```

Some of the steps of this shell script will take a bit of time, in the order of several minutes. Hang on! When the command is done, the file *source.input* is ready for use for by *abstfinderSummary* or *abstfinderFull*. Generally, one uses the former as it gives enough information for the user who is moderately familiar with the input to find the abstractions.

4. Assuming that we are using *abstfinderSummary* on the file *source.input* do the following:

```
abstfinderSummary source.input source.output.
```

This step will take a *long* time, like an hour and a quarter for the average technical paper. So go out to lunch and come back sated to see the output! Besides generating the list of abstractions together with pointers for finding the phrases in the input, *abstfinderSummary* also creates the file *CorrPhrases* that lists only the abstractions. This is intended to be used for straining. If a third argument is given in the invocation of *abstfinderSummary*, that argument is used as the file name for this list of only the abstractions.

5. Now you must examine *source.output* to determine what is to be ignored as an application-particular term for the next run. Any such terms can be put into the file *IgnoredApplication*. Then the above steps are repeated. At some point, the user is satsified that all the generate terms above a certain point (identified by the user) are abstractions.

6. If you wish to check that there is no new significant information below the certain point, he or she uses the strainer to filter out what is known to be significant and then runs *abstfinderSummary* or *abstfinderFull* on what is left to see that it does not yield anything significant. The user must first edit *CorrPhrases* to delete the significant phrases below the identified certain point. Then *strainTermsFromInput* is invoked.

```
strainTermsFromInput source.cr source.strain
```

In the continuing example, *source.cr* is the input file to be strained, and *source.strain* is the output file, to be used as input to *abstfinderSummary* in Step 7. The program *strainTermsFromInput* uses the edited *CorrPhrases* file.

7. Now *abstfinderSummary* can be invoked for the next iteration. Notice that in the continuing example, you have to use *source.strain* as the input file to *abstfinderSummary*. Notice also that both *abstfinderSummary* and *abstfinderFull* write over *CorrPhrases* on each run. Therefore, it is useful to have the third argument for *abstfinderSummary* or *abstfinderFull*.

FILES

IgnoredFile	general terms to ignore
IgnoredApplication	application specific terms to ignore
IgnoredSuffixes	suffixes to ignore
CorrPhrases	just the abstractions

SEE ALSO

makeSentencesToLines(1), **eliminateIgnoredParts**(1), **abstfinderSummary**(1), **abstfinderFull**(1), **strainTermsFromInput**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

makeSentencesToLines □ reorganize input to put each sentence in its own line.

SYNOPSIS

makeSentencesToLines *infile outfile*

DESCRIPTION

makeSentencesToLines is a tool of *AbstFinder* for producing a list of identified abstractions from ASCII file written in any natural language or any formal language (programming or other).

makeSentencesToLines is a shell script, invoking *newcr*(1) and *tr*(1), that reorganizes the contents of *infile* into lines, each containing one sentence according to a pre-defined punctuation symbols list to yield *outfile*.

The list of punctuation symbols marking the ends of sentences is defined as a shell variable inside the script; the list can easily be edited to suit the user's needs.

SEE ALSO

AbstFinder(1), **newcr**(1), **tr**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

eliminateIgnoredParts — remove ignored terms from an ASCII file

SYNOPSIS

eliminateIgnoredParts *infile outfile*

DESCRIPTION

eliminateIgnoredParts is a tool of *AbstFinder* for producing a list of identified abstractions from ASCII file written in any natural language or any formal language (programming or other).

It is actually a shell script for removing from *infile* the ignored words listed in *IgnoredFile*, *IgnoredApplication*, and *IgnoredSuffixes*. The result goes to *outfile*.

First, *eliminateIgnoredParts* changes all characters to lower case, using *tr*(1). Then, it activates the *input* program in order to remove the ignored words.

SEE ALSO

AbstFinder(1), **input**(1), **tr**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

abstfinderFull □ find abstractions in ASCII text

SYNOPSIS

abstfinderFull *infile outfile* [*optional_new_name_for_CorrPhrases*]

DESCRIPTION

abstfinderFull is a tool of *AbstFinder* for producing a list of identified abstractions from ASCII file written in any natural language or any formal language (programming or other).

abstfinderFull is actually a shellscript that uses *abstfinder.ful* to identify the abstractions in *infile* and to output them to *outfile*. The output of *AbstFinder* is in two parts. The first part is a summary of the identified abstractions, and the second part contains a full description of the abstractions. The summary lists the abstractions and gives pointers (sentence numbers) to where they are found in *input*. The full description shows the sentences containing the abstractions.

As a side effect, *abstfinder.ful* generates a file *CorrPhrases* containing the summary without the pointers. If a third argument is provided to *abstfinderFull*, then *CorrPhrases* is renamed to have the third argument as its name. It is recommended to use this third argument for all runs but the one whose input and output are submitted to *strainTermsFromInput*.

FILES

CorrPhrases

SEE ALSO

AbstFinder(1), **abstfinder.ful**(1) **strainTermsFromInput**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

abstfinderSummary □ find abstractions in ASCII text

SYNOPSIS

abstfinderSummary *infile outfile* [*optional_new_name_for_CorrPhrases*]

DESCRIPTION

abstfinderSummary is a tool of *AbstFinder* for producing a list of identified abstractions from ASCII file written in any natural language or any formal language (programming or other).

abstfinderSummary is actually a shellscript that uses *abstfinder.sum* to identify the abstractions in *infile* and to output them to *outfile*. The output of *AbstFinder* is a summary of the identified abstractions. This is the same as the first part of the output generated by *abstfinderFull*. This summary lists the abstractions and gives pointers (sentence numbers) to where they are found in *input*.

As a side effect, *abstfinder.ful* generates a file *CorrPhrases* containing the summary without the pointers. If a third argument is provided to *abstfinderSummary*, then *CorrPhrases* is renamed to have the third argument as its name. It is recommended to use this third argument for all runs but the one whose input and output are submitted to *strainTermsFromInput*.

FILES

CorrPhrases

SEE ALSO

AbstFinder(1), **abstfinder.sum**(1) **abstfinderFull**(1) **strainTermsFromInput**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

strainTermsFromInput □ strain terms from an ASCII file.

SYNOPSIS

strainTermsFromInput *infile outfile*

DESCRIPTION

strainTermsFromInput is tool of *AbstFinder* for producing a list of identified abstractions from ASCII file written in any natural language or any formal language (programming or other).

strainTermsFromInput is actually a shell script for removing from *infile* all terms found in the files *CorrPhrases* and *IgnoredApplication*. The result is sent to *outfile*.

First, it changes all characters to lower case, using *tr*. Then, it activates the *strainer* program to do the actual straining.

FILES

CorrPhrases

SEE ALSO

AbstFinder(1), **strainer**(1), **tr**(1),

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

abstfinder.ful □ find abstractions

SYNOPSIS

abstfinder.ful *infile outfile*

DESCRIPTION

abstfinder.ful identifies abstractions from *infile* and generates the output to *outfile*. The output of *abstfinder* is in two parts. The first part is a summary of the identified abstractions together with pointers (sentence numbers) to where in *infile* they are found, and the second part contains a full description of the abstractions.

abstfinder.ful also creates a file *CorrPhrases* that lists only the abstractions (the first part without the pointers). This file is intended to be used as input to *strainTermsFromInput*.

FILES

CorrPhrases

SEE ALSO

AbstFinder(1), **abstfinderFull**(1), **strainTermsFromInput**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

abstfinder.sum □ find abstractions

SYNOPSIS

abstfinder.sum *infile outfile*

DESCRIPTION

abstfinder.sum identifies abstractions from *infile* and generates the output to *outfile*. The output of *abstfinder* is identical to that of the first part of *abstfinder.ful*(1) and is a summary of the identified abstractions together with pointers (sentence numbers) to where in *infile* they are found.

abstfinder.sum also creates a file *CorrPhrases* that lists only the abstractions (*outfile* without the pointers). This file is intended to be used as input to *strainTermsFromInput*.

FILES

CorrPhrases

SEE ALSO

AbstFinder(1), **abstfinderSummary**(1), **abstfinder.ful**(1), **strainTermsFromInput**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

strainer − strain terms from an ASCII file.

SYNOPSIS

strainer *infile outfile* [**−i**]

DESCRIPTION

strainer is a program that is activated by *strainTermsFromInput* which is a tool of *AbstFinder* for producing list of identified abstractions from ASCII file written in any natural language or any formal language (programming or other).

strainer removes from *infile* all terms taken from *CorrPhrases* and, if **−i** was mentioned in the command line, from *IgnoredApplication* and sends the result to *outfile*.

CorrPhrases is a file generated by *abstfinderFull* or *abstfinderSummary*.

FILES

CorrPhrases

SEE ALSO

AbstFinder(1), **abstfinderFull**(1), **abstfinderSummary**(1), **strainTermsFromInput**(1),

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

input □ remove ignored terms from an ASCII file

SYNOPSIS

input *infile outfile*

DESCRIPTION

input is a program that is activated by *eliminateIgnoredParts*, which is a tool of *AbstFinder* for producing a list of identified abstractions from ASCII file written in any natural language or any formal language (programming or other).

input removes from *infile* words found in *IgnoredFile*, *IgnoredApplication*, and *IgnoredSuffixes*, and sends the result to *outfile*. Note that while the ignored words are completely filtered, the ignored suffixes are marked only in their eighth bits in order to be able to ignore them in the process of abstraction identification.

SEE ALSO

AbstFinder(1), **eliminateIgnoredParts**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

NAME

newcr □ replace two successive CRs with a dot and remove all redundant CRs.

SYNOPSIS

newcr

DESCRIPTION

newcr is a filter used by *makeSentencesToLines* to replace two successive CRs with a dot and to remove all redundant CRs, As a filter, the program accepts input from the standard input and outputs to the standard output.

SEE ALSO

AbstFinder(1), **makeSentencesToLines**(1)

AUTHOR

Leah Goldin, Computer Science Department, Technion, Haifa, Israel.

Appendix C Results and Data of the findphrases Case Study

This appendix contains AbstFinder data and results on the findphrases case study:

1. Section C.1 contains the man page of findphrases that was used as the specification document in this case study.

2. Section C.2 contains the ignored files that were used by AbstFinder.

3. Section C.3 contains AbstFinder generated output list of abstractions.

4. Section C.4 contains already known abstractions by Aguilera, the designer of findphrases.

C.1 Source Transcript of the findphrases Case Study

The manual pages for findphrases are in UNIX manual page format on the subsequent pages. Their page numbers have been edited to match their position in this thesis. The input to AbstFinder for this case study was an ASCII version of the formatted manual page.

NAME

findphrases find repeated phrases in an arbitrary text

SYNOPSIS

findphrases [**n***number*] **p***punctuation-keyword-file* [**x***ignored-phrases-file*] [**m***multi-tokens-file*] [**u**] [**b**] [**s**] [**t**] [**v**] [**c**]

DESCRIPTION

All files mentioned in the synopsis provide their data in what is referred to as free format subject to particular restrictions to be described for each case. In free format, the items of the file may be entered zero or several per line with a mixture of blanks and tabs before, in between, and after the items. Consequently, no item can include a blank, a tab, or a newline.

The **n** argument is optional and if present provides a number *number* serving as the maximum length phrase (to be described later) to be tallied. If this argument is not present, if it does not supply a number, or if the supplied number is outside the reasonable range of greater than zero and less than or equal to 50, then *number* is taken as 10.

The *punctuation-keyword-file* contains in free format a list of those character strings to be taken as punctuation/keywords (see below). The optional *ignored-phrases-file* contains one-per-line a list of phrases to be ignored in the tallying (see below). In each line, the tokens (see below) are in free format. The optional *multi-tokens-file* contains in free format a list of those character strings consisting of more than one symbolcharacter (see below) which are to be taken as multi-tokens (see below).

No assumptions are made about the standard input, thus it may be an arbitrary text. The program parses the text into words and symbolcharacters. These in turn are formed and classified into tokens and punctuation/keywords based on the information provided by the *punctuation-keyword-file* and, when the **m** option is present, the *multi-tokens-file*.

First some definitions are necessary:

Whitespace: blank, *tab*, *newline* ,.IR

Wordcharacter: letter, *digit* ,.B_

Symbolcharacter: any printable character which is neither a wordcharacter nor a blank

Word: any sequence of wordcharacters delimited on each side by whitespace or a symbolcharacter

Punctuation/Keyword: whatever is in the *punctuation-keyword-file*; the symbolcharacter strings are called punctuation and the wordcharacter strings are called keywords

Multi-token: whatever is in the *multi-tokens-file*

Token: any word, symbolcharacter, or multi-token which is not listed in the *punctuation-keyword-file*

Sentence: list of tokens delimited on each side by punctuation/keyword

Phrase: one or more consecutive tokens occurring within one sentence

The main job of this program is to tally the occurrence of all phrases in all sentences. The maximum length phrase that has to be considered is that of *number* tokens. If the *ignored-phrases-file* is provided, then the phrases given in the file are to be ignored in the tallying. If the **b** option is used along with the *ignored-phrases-file,* then phrases which begin with an ignored phrase are also ignored in the tallying.

The standard output consists of:

a copy of the input as is, with the lines numbered and the punctuation/keywords overstruck two times (i.e., printed three times in place) so that they can be spotted easily,

a frequency ranked table of the repeated phrases. i.e., those appearing more than once among the sentences; that is the entries of the table are given in order of decreasing frequency, and

an alphabetically ordered table of the repeated phrases.

In the two tables, the entry for a repeated phrase consists of:

a sequence of asterisks indicating the phrase's frequency as a percentage of the maximum frequency; in this one asterisk represents 10%,

the actual number of occurrences of the repeated phrase,

the repeated phrase itself, and

a list of the numbers of all lines containing the beginning of the repeated phrase.

In printing the repeated phrase itself in a table entry, the underscores, i.e., ''_'', are printed as blanks. This means that an underscore can be used immediately preceeding or following a word that looks like a keyword to prevent it from being considered a keyword.

Note that the definition of ''phrase'' is independent of the number of times it occurs in the sentences. An *ignored phrase* is simply one to be ignored in the tallying but not in searching for phrases. A phrase which contains an ignored phrase which itself is not ignored is to be tallied. When the **-b** option is present, a phrase which begins with an ignored phrase is not to be tallied. A *repeated phrase* is one whose final tally is greater than one. Only the repeated phrases show up in the tables of the output.

Typically, the *ignored-phrases-file* will contain so-called noise phrases such as ''a'', ''an'', ''the'', ''of'', ''of the'', etc. plus any useless phrases found in previous runs of the program.

One particular configuration of the files is as follows:

Punctuation-keyword-file: **; [ ] abort accept access all and array at begin body case constant declare delta digits do else elsif end entry exception exit for function generic goto if in is limited loop mod new not null of or others out package pragma private procedure raise range record rem renames return reverse select separate subtype task terminate then type use when while with xor**

Multi-tokens-file: **-- := <= .B**

This configuration is suited for finding repeated phrases in Ada™ (Ada is a trademark of the U. S. Department of Defense.) or in an Ada-based program design language.

If the **-u** option is present, then only the unique phrases that are not wholly and everywhere contained in another phrase are listed in the tables of the output. In addition to the already specified output, if the **-s** option is present, then all the sentences are listed; if the **-t** option is present, then all the tokens are listed; if the **-v** option is present, then the output is verbose with the punctuation/keywords listed, and when the **-m**, and respectively the **-x**, option is present, the multi-tokens, and respectively the ignored phrases, are listed. If the **-c** option is present, then upper and lower case distinctions are to be applied in determining whether a phrase is in a sentence. The default is to ignore case distinction in the comparisons.

DIAGNOSTICS

They are good, of course.

BUGS

There are none, of course.

C.2 Ignored Files Used in the findphrases Case Study

ignored-common-word-file:

```
the of is to an a from and as can be be being do does done did with very high
later well use using used in or at on want wanted anywhere until many new
somewhere whatever will for not by should able all have has had when more than
top are how first thereby any would make made give through this here between
within everywhere end zero underscore
two when generally immediately only now sufficient after continued case cases
shall initial that begin gegun successful take taken these both unique also
low one normal without possible ready kind most during just anything talk
about it go went up try tight somthing if i into move moves moved out get
getting like each provide provides contains contained containing particular
before allways back send sent you fact second third whether its but what same
other straight where me sure how we depend actually onto affect necessarily
necessary could very else instance they see side even matter told him said
off no so three several those right since some another which alright set
put much correct synopsis include item items
becuase was there then something under even effective affect available allow
thing range consist consisting maximum length contain contains
given less greater consequently below itself ever per
```

ignored-application-words-file:

```
findphrases ignored ignore repeated repeat
distinction program  terminate
present described provided begin beginning print printed printing printable
call called standard course entry occur occurring occurrs
occurrence occurrences number numbers definition
definitions times considered form turn based frequency follow follows following
ordered sign limited vers sequence represents
tally tallied tallying
```

ignored-suffixes-file:

```
ition ntion ption nction ction ation tion
ance ence ment sent able ered ting ings ing
sist tive pplied ified lied  ted ited ied ed
ically ally lly
```

C.3 AbstFinder Results in the findphrases Case Study

In this and other appendix sections showing AbstFinder output, long lines in part 1 of the AbstFinder output are truncated by their extending beyond the right-hand margin of the page. Despite the ugliness of the truncation, it corresponds to the reality of the user ignoring the too much detail that a too long line represents. In any case, that which is missing in the truncation can be seen by looking in part 2 of the AbstFinder output.

```
        ***************************************************
        S U M M A R Y   OF      A B S T F I N D
        ***************************************************

-------------------------------------------------------
# of lines read from input file is 54
# of abstractions found is 49
-------------------------------------------------------

------  |------ |------ |------ |---------------------
 #)     | abst# | corr_ | corr_ | correlateted-phrases
        |       |phras# |lines# |
------  |------ |------ |------ |---------------------
 1      | 42    | 1     | 11    |  punctuation keyword file|
 2      | 14    | 1     | 2     |  whitespace|
 3      | 6     | 1     | 2     | argument |
 4      | 38    | 2     | 25    |  phrase| phrase |
 5      | 44    | 2     | 14    | s file|tokens file|
 6      | 23    | 2     | 12    | tokens |s sentence|
 7      | 16    | 2     | 9     | character|symbolcharacter|
 8      | 43    | 2     | 8     | multi |r multi |
 9      | 2     | 2     | 6     |  free format |files |
 10     | 4     | 2     | 5     |  blank| blank tab newline|
 11     | 41    | 2     | 3     | files | configuration |
 12     | 11    | 2     | 3     | arbitrary text|input |
 13     | 26    | 3     | 29    |  phrases |file phrases file | phrases file phrases |
 14     | 5     | 4     | 28    | phrase|argument |optional | phrase |
 15     | 24    | 4     | 26    | phrases |s sentence| phrase|s sentences|
 16     | 45    | 4     | 26    | phrases a| phrase| phrases | configuration |
 17     | 20    | 4     | 17    | s file | multi tokens file|e token| multi tokens file |
 18     | 33    | 4     | 12    | keyword|keyword |keyword p|d prev|
 19     | 12    | 4     | 12    | character|words |symbolcharacter|characters|
 20     | 25    | 5     | 31    |  phrase|tokens| phrase |e tokens| phrase t|
 21     | 0     | 5     | 26    | phrases |arbitrary text|d phrase| phrases |phrases a|
 22     | 29    | 5     | 25    | phrases| phrase| phrases|table phrases| table|
 23     | 35    | 5     | 25    | phrases| phrase| phrase | phrases| phrase s|
 24     | 36    | 5     | 25    |  phrase|e phrase|phrase | phrase | phrase phrase |
```

```
25    | 39    | 5     | 25    |  phrases |output| tables | phrases s|tables output|
26    | 9     | 5     | 14    | tokens f|free format| line |e token|e tokens|
27    | 15    | 5     | 11    |  blank| blank tab newline|file f|character|wordcharacter|
28    | 17    | 5     | 10    |  blank |character | character |wordcharacter|wordcharacter |
29    | 3     | 5     | 10    | file m| free format | blank| line |blanks|
30    | 21    | 6     | 20    |  punctuation keyword file |multi token|character |symbolcharacter n
31    | 22    | 7     | 37    | d phrase|punctuation keyword |tokens | list |s delimited |keyword p
32    | 48    | 7     | 28    | phrase|phrase |option| option| option |e sentence| phrase |
33    | 34    | 7     | 26    | phrase|e phrase|phrase |s sentence|s sentences| phrase |e phrase |
34    | 30    | 7     | 25    | phrase|phrase | phrase |s phrase | table|consists | tables |
35    | 7     | 7     | 19    |  punctuation keyword file |free format | free format |file free for
36    | 32    | 8     | 27    | phrase| phrase|blanks| blank| phrase | phrase t|table | table|
37    | 37    | 8     | 27    | phrase|phrase |option|option | phrase | b option |s phrase |option
38    | 8     | 9     | 30    | phrases |phrases file |optional | line | optional | phrase| phrases
39    | 18    | 9     | 17    |  punctuation keyword|character |symbolcharacter |characters| whites
40    | 1     | 10    | 39    | phrases |file m| punctuation keyword file |phrases file |tokens f|n
41    | 40    | 10    | 30    | phrases |e phrases |phrases f|s file|phrases file | phrases file |s
42    | 19    | 11    | 18    |  punctuation keyword file |multi token|keywords |character strings
43    | 28    | 12    | 34    | phrases| punctuation keyword|punctuation keywords |input | phrase|
44    | 27    | 12    | 29    | phrases |phrases file |file phrases |phrase |option| option|option
45    | 31    | 12    | 25    | phrase|e phrase|phrase |e list |al phrase| phrase |es phrase|t line
46    | 47    | 13    | 39    | phrases |tokens | punctuation keyword|multi tokens |option|punctuat
47    | 10    | 13    | 29    | multi tokens |multi tokens file|s file |free format | free format |
48    | 13    | 13    | 28    |  punctuation keyword|file m|multi tokens file| punctuation keyword
```

```
        ************************
        abstractions found
        ************************

--------------------------------------------------
{1} abst_id=42
==================
correlated phrases of abstraction are
(#=1)
         punctuation keyword file|

correlated sentences of abstraction are
(#=11)
         44 2 8 14 20 21 23 24 30 35 49
--------------------------------------------------
{2} abst_id=14
==================
correlated phrases of abstraction are
(#=1)
         whitespace|

correlated sentences of abstraction are
(#=2)
         16 20
--------------------------------------------------
{3} abst_id=6
==================
correlated phrases of abstraction are
(#=1)
        argument |

correlated sentences of abstraction are
(#=2)
         7 6
--------------------------------------------------
{4} abst_id=38
==================
correlated phrases of abstraction are
(#=2)
         phrase| phrase |

correlated sentences of abstraction are
(#=25)
         40 1 2 6 9 24 26 27 28 29 30 31 32 33 34 36
         37 38 39 41 42 47 48 49 50
```

```
-------------------------------------------------
{5} abst_id=44
==================
correlated phrases of abstraction are
(#=2)
        s file|tokens file|

correlated sentences of abstraction are
(#=14)
         46 2 9 10 11 14 22 24 25 27 28 29 42 49
-------------------------------------------------
{6} abst_id=23
==================
correlated phrases of abstraction are
(#=2)
        tokens |s sentence|

correlated sentences of abstraction are
(#=12)
         25 2 10 11 14 22 24 26 27 36 46 49
-------------------------------------------------
{7} abst_id=16
==================
correlated phrases of abstraction are
(#=2)
        character|symbolcharacter|

correlated sentences of abstraction are
(#=9)
         18 8 11 13 17 19 20 21 23
-------------------------------------------------
{8} abst_id=43
==================
correlated phrases of abstraction are
(#=2)
        multi |r multi |

correlated sentences of abstraction are
(#=8)
         45 2 11 14 21 22 23 49
-------------------------------------------------
{9} abst_id=2
==================
correlated phrases of abstraction are
(#=2)
```

```
         free format |files |

correlated sentences of abstraction are
(#=6)
         3 4 8 10 11 43
-------------------------------------------------
{10} abst_id=4
==================
correlated phrases of abstraction are
(#=2)
         blank| blank tab newline|

correlated sentences of abstraction are
(#=5)
         5 4 17 19 34
-------------------------------------------------
{11} abst_id=41
==================
correlated phrases of abstraction are
(#=2)
        files | configuration |

correlated sentences of abstraction are
(#=3)
         43 3 47
-------------------------------------------------
{12} abst_id=11
==================
correlated phrases of abstraction are
(#=2)
        arbitrary text|input |

correlated sentences of abstraction are
(#=3)
         12 1 30
-------------------------------------------------
{13} abst_id=26
==================
correlated phrases of abstraction are
(#=3)
         phrases |file phrases file | phrases file phrases

correlated sentences of abstraction are
(#=29)
         28 1 2 6 9 11 14 22 24 26 27 29 30 31 32 33
```

```
         34 36 37 38 39 40 41 42 46 47 48 49 50
-------------------------------------------------
{14} abst_id=5
==================
correlated phrases of abstraction are
(#=4)
        phrase|argument |optional | phrase |

correlated sentences of abstraction are
(#=28)
         6 1 2 7 9 11 14 24 26 27 28 29 30 31 32 33
         34 36 37 38 39 40 41 42 47 48 49 50
-------------------------------------------------
{15} abst_id=24
==================
correlated phrases of abstraction are
(#=4)
        phrases |s sentence| phrase|s sentences|

correlated sentences of abstraction are
(#=26)
         26 1 2 6 9 24 25 27 28 29 30 31 32 33 34 36
         37 38 39 40 41 42 47 48 49 50
-------------------------------------------------
{16} abst_id=45
==================
correlated phrases of abstraction are
(#=4)
        phrases a| phrase| phrases | configuration

correlated sentences of abstraction are
(#=26)
         47 1 2 6 9 24 26 27 28 29 30 31 32 33 34 36
         37 38 39 40 41 42 43 48 49 50
-------------------------------------------------
{17} abst_id=20
==================
correlated phrases of abstraction are
(#=4)
        s file | multi tokens file|e token| multi tokens file

correlated sentences of abstraction are
(#=17)
         22 2 9 10 11 14 21 23 24 25 27 28 29 42 45 46
         49
```

```
-------------------------------------------------
{18} abst_id=33
==================
correlated phrases of abstraction are
(#=4)
        keyword|keyword |keyword p|d prev|

correlated sentences of abstraction are
(#=12)
         35 2 8 14 20 21 23 24 30 42 44 49
-------------------------------------------------
{19} abst_id=12
==================
correlated phrases of abstraction are
(#=4)
        character|words |symbolcharacter|characters

correlated sentences of abstraction are
(#=12)
         13 8 11 14 17 18 19 20 21 23 30 49
-------------------------------------------------
{20} abst_id=25
==================
correlated phrases of abstraction are
(#=5)
         phrase|tokens| phrase |e tokens| phrase t

correlated sentences of abstraction are
(#=31)
         27 1 2 6 9 10 11 14 22 24 25 26 28 29 30 31
         32 33 34 36 37 38 39 40 41 42 46 47 48 49 50
-------------------------------------------------
{21} abst_id=0
==================
correlated phrases of abstraction are
(#=5)
        phrases |arbitrary text|d phrase| phrases
        phrases a|

correlated sentences of abstraction are
(#=26)
         1 2 6 9 12 24 26 27 28 29 30 31 32 33 34 36
         37 38 39 40 41 42 47 48 49 50
-------------------------------------------------
{22} abst_id=29
```

```
==================
correlated phrases of abstraction are
(#=5)
        phrases| phrase| phrases|table phrases| table

correlated sentences of abstraction are
(#=25)
         31 1 2 6 9 24 26 27 28 29 30 32 33 34 36 37
         38 39 40 41 42 47 48 49 50
--------------------------------------------------
{23} abst_id=35
==================
correlated phrases of abstraction are
(#=5)
        phrases| phrase| phrase | phrases| phrase s

correlated sentences of abstraction are
(#=25)
         37 1 2 6 9 24 26 27 28 29 30 31 32 33 34 36
         38 39 40 41 42 47 48 49 50
--------------------------------------------------
{24} abst_id=36
==================
correlated phrases of abstraction are
(#=5)
         phrase|e phrase|phrase | phrase | phrase phrase

correlated sentences of abstraction are
(#=25)
         38 1 2 6 9 24 26 27 28 29 30 31 32 33 34 36
         37 39 40 41 42 47 48 49 50
--------------------------------------------------
{25} abst_id=39
==================
correlated phrases of abstraction are
(#=5)
         phrases |output| tables | phrases s|tables output

correlated sentences of abstraction are
(#=25)
         41 1 2 6 9 24 26 27 28 29 30 31 32 33 34 36
         37 38 39 40 42 47 48 49 50
--------------------------------------------------
{26} abst_id=9
==================
```

```
correlated phrases of abstraction are
(#=5)
        tokens f|free format| line |e token|e tokens

correlated sentences of abstraction are
(#=14)
         10 2 3 4 8 9 11 14 22 24 25 27 46 49
-------------------------------------------------
{27} abst_id=15
==================
correlated phrases of abstraction are
(#=5)
         blank| blank tab newline|file f|character
        wordcharacter|

correlated sentences of abstraction are
(#=11)
         17 4 5 8 11 13 18 19 20 21 23
-------------------------------------------------
{28} abst_id=17
==================
correlated phrases of abstraction are
(#=5)
         blank |character | character |wordcharacter
        wordcharacter |

correlated sentences of abstraction are
(#=10)
         19 5 8 11 13 17 18 20 21 23
-------------------------------------------------
{29} abst_id=3
==================
correlated phrases of abstraction are
(#=5)
        file m| free format | blank| line |blanks

correlated sentences of abstraction are
(#=10)
         4 2 3 5 8 10 11 14 17 34
-------------------------------------------------
{30} abst_id=21
==================
correlated phrases of abstraction are
(#=6)
         punctuation keyword file |multi token|character
```

```
        symbolcharacter multi token|listed |e sentence

correlated sentences of abstraction are
(#=20)
         23 2 8 11 13 14 17 18 19 20 21 22 24 30 35 44
         45 48 49 50
--------------------------------------------------
{31} abst_id=22
==================
correlated phrases of abstraction are
(#=7)
        d phrase|punctuation keyword |tokens | list
        s delimited |keyword p| punctuation keyword

correlated sentences of abstraction are
(#=37)
         24 1 2 6 8 9 10 11 14 20 21 22 23 25 26 27
         28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 44
         46 47 48 49 50
--------------------------------------------------
{32} abst_id=48
==================
correlated phrases of abstraction are
(#=7)
        phrase|phrase |option| option| option |e sentence
         phrase |

correlated sentences of abstraction are
(#=28)
         50 1 2 6 9 11 14 23 24 26 27 28 29 30 31 32
         33 34 36 37 38 39 40 41 42 47 48 49
--------------------------------------------------
{33} abst_id=34
==================
correlated phrases of abstraction are
(#=7)
        phrase|e phrase|phrase |s sentence|s sentences
         phrase |e phrase |

correlated sentences of abstraction are
(#=26)
         36 1 2 6 9 24 25 26 27 28 29 30 31 32 33 34
         37 38 39 40 41 42 47 48 49 50
--------------------------------------------------
{34} abst_id=30
```

```
==================
correlated phrases of abstraction are
(#=7)
        phrase|phrase | phrase |s phrase | table|consists
         tables |

correlated sentences of abstraction are
(#=25)
         32 1 2 6 9 24 26 27 28 29 30 31 33 34 36 37
         38 39 40 41 42 47 48 49 50
--------------------------------------------------
{35} abst_id=7
==================
correlated phrases of abstraction are
(#=7)
         punctuation keyword file |free format | free format
        file free format list character strings |words
        punctuation keywords |character strings punctuation

correlated sentences of abstraction are
(#=19)
         8 2 3 4 10 11 13 14 17 18 19 20 21 23 24 30
         35 44 49
--------------------------------------------------
{36} abst_id=32
==================
correlated phrases of abstraction are
(#=8)
        phrase| phrase|blanks| blank| phrase | phrase t
        table | table|

correlated sentences of abstraction are
(#=27)
         34 1 2 4 5 6 9 24 26 27 28 29 30 31 32 33
         36 37 38 39 40 41 42 47 48 49 50
--------------------------------------------------
{37} abst_id=37
==================
correlated phrases of abstraction are
(#=8)
        phrase|phrase |option|option | phrase | b option
        s phrase |option phrase|

correlated sentences of abstraction are
(#=27)
```

```
         39 1 2 6 9 11 14 24 26 27 28 29 30 31 32 33
         34 36 37 38 40 41 42 47 48 49 50
--------------------------------------------------
{38} abst_id=8
==================
correlated phrases of abstraction are
(#=9)
        phrases |phrases file |optional | line | optional
         phrase| phrases file |e list |al phrase|

correlated sentences of abstraction are
(#=30)
         9 1 2 6 10 11 14 22 24 26 27 28 29 30 31 32
         33 34 36 37 38 39 40 41 42 46 47 48 49 50
--------------------------------------------------
{39} abst_id=18
==================
correlated phrases of abstraction are
(#=9)
         punctuation keyword|character |symbolcharacter
        characters| whitespace|wordcharacter| wordcharacter
        e symbolcharacter |s delimited |

correlated sentences of abstraction are
(#=17)
         20 2 8 11 13 14 16 17 18 19 21 23 24 30 35 44
         49
--------------------------------------------------
{40} abst_id=1
==================
correlated phrases of abstraction are
(#=10)
        phrases |file m| punctuation keyword file
        phrases file |tokens f|multi tokens |multi tokens file
         multi tokens file| phrase|file phrases file

correlated sentences of abstraction are
(#=39)
         2 1 4 6 8 9 10 11 14 20 21 22 23 24 25 26
         27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42
         44 45 46 47 48 49 50
--------------------------------------------------
{41} abst_id=40
==================
correlated phrases of abstraction are
```

```
(#=10)
        phrases |e phrases |phrases f|s file|phrases file
         phrases file |s phrase|se phrase|d prev| phrases s

correlated sentences of abstraction are
(#=30)
         42 1 2 6 9 11 14 22 24 26 27 28 29 30 31 32
         33 34 35 36 37 38 39 40 41 46 47 48 49 50
--------------------------------------------------
{42} abst_id=19
==================
correlated phrases of abstraction are
(#=11)
         punctuation keyword file |multi token|keywords
        character strings |character strings punctuation
        symbolcharacter |wordcharacter|wordcharacter
         wordcharacter|e symbolcharacter | multi token

correlated sentences of abstraction are
(#=18)
         21 2 8 11 13 14 17 18 19 20 22 23 24 30 35 44
         45 49
--------------------------------------------------
{43} abst_id=28
==================
correlated phrases of abstraction are
(#=12)
        phrases| punctuation keyword|punctuation keywords
        input | phrase| phrases|table phrases| table
        consists |t lines |output| output |

correlated sentences of abstraction are
(#=34)
         30 1 2 6 8 9 12 13 14 20 21 23 24 26 27 28
         29 31 32 33 34 35 36 37 38 39 40 41 42 44 47 48
         49 50
--------------------------------------------------
{44} abst_id=27
==================
correlated phrases of abstraction are
(#=12)
        phrases |phrases file |file phrases |phrase
        option| option|option | phrase | phrases file phrases
        s phrase |es phrase| b option |
```

```
correlated sentences of abstraction are
(#=29)
         29 1 2 6 9 11 14 22 24 26 27 28 30 31 32 33
         34 36 37 38 39 40 41 42 46 47 48 49 50
-------------------------------------------------
{45} abst_id=31
==================
correlated phrases of abstraction are
(#=12)
        phrase|e phrase|phrase |e list |al phrase
         phrase |es phrase|t lines |e phrase | phrase s
         phrase phrase |phrase lis|

correlated sentences of abstraction are
(#=25)
         33 1 2 6 9 24 26 27 28 29 30 31 32 34 36 37
         38 39 40 41 42 47 48 49 50
-------------------------------------------------
{46} abst_id=47
==================
correlated phrases of abstraction are
(#=13)
        phrases |tokens | punctuation keyword|multi tokens
        option|punctuation keywords | option|option multi tokens
        listed | phrase| phrases | output | option

correlated sentences of abstraction are
(#=39)
         49 1 2 6 8 9 10 11 13 14 20 21 22 23 24 25
         26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41
         42 44 45 46 47 48 50
-------------------------------------------------
{47} abst_id=10
==================
correlated phrases of abstraction are
(#=13)
        multi tokens |multi tokens file|s file |free format
         free format |optional |file free format list character strings
         optional |symbolcharacter|multi tokens file
        symbolcharacter | multi tokens file |symbolcharacter multi token

correlated sentences of abstraction are
(#=29)
         11 2 3 4 6 8 9 10 13 14 17 18 19 20 21 22
         23 24 25 27 28 29 39 42 45 46 48 49 50
```

```
--------------------------------------------------
{48} abst_id=13
==================
correlated phrases of abstraction are
(#=13)
         punctuation keyword|file m|multi tokens file
         punctuation keyword file |s file |option
        punctuation keywords | option|multi tokens file
         multi tokens file |option | option |option multi tokens

correlated sentences of abstraction are
(#=28)
         14 2 4 6 8 9 10 11 13 20 21 22 23 24 25 27
         28 29 30 35 39 42 44 45 46 48 49 50
--------------------------------------------------
{49} abst_id=46
==================
correlated phrases of abstraction are
(#=14)
        phrases |option|phrase | option| option |listed
         phrase| phrase | phrases |output| tables
        phrase lis|option phrase|tables output|

correlated sentences of abstraction are
(#=28)
         48 1 2 6 9 11 14 23 24 26 27 28 29 30 31 32
         33 34 36 37 38 39 40 41 42 47 49 50
--------------------------------------------------
```

C.4 Decomposition of the findphrases Case Study

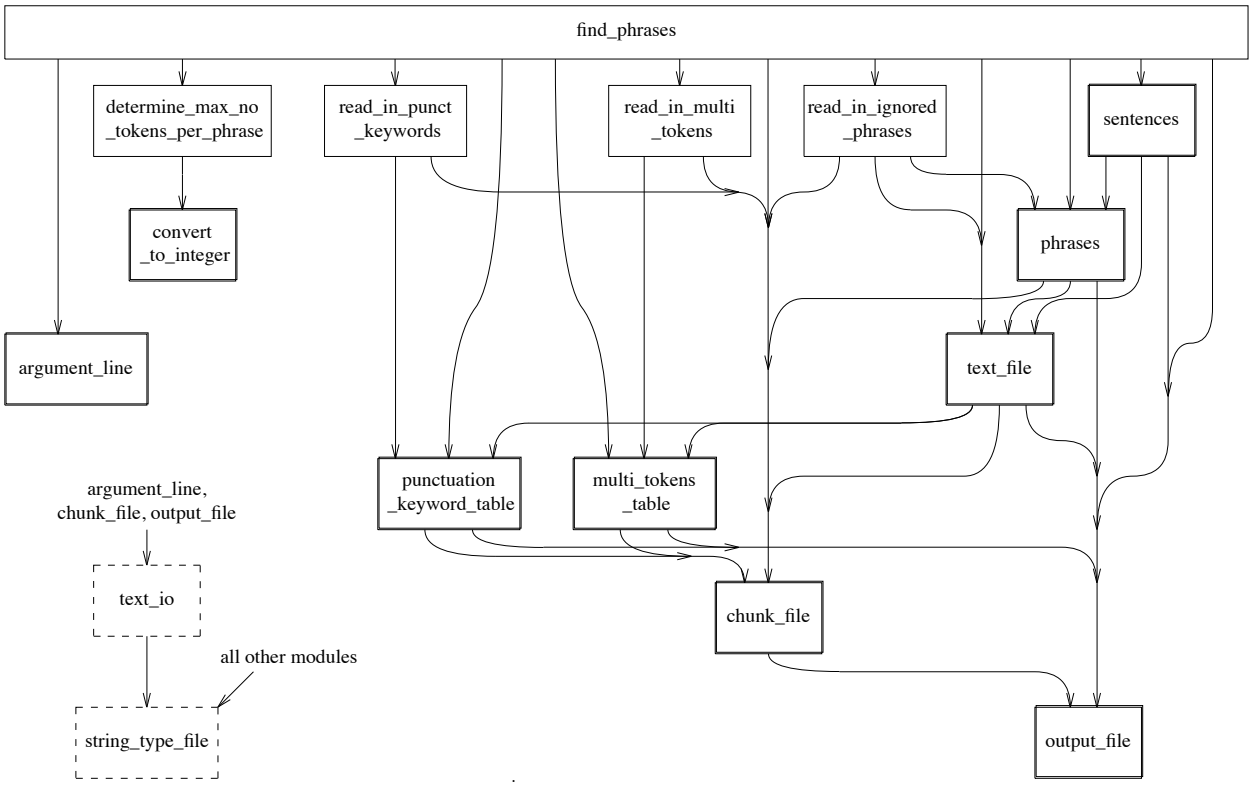


Figure 14: Decomposition of findphrases by Aguilera
Aguilera ציור 14:פירוק של findphrases על ידי

Appendix D Results and Data of the Flinger Missile Case Study

This appendix contains AbstFinder data and results of the Flinger Missile case study:

1. Section D.1 contains the specification document of the Flinger Missile that was used as the specification document in this case study.

2. Section D.2 contains the ignored files that were used by AbstFinder.

3. Section D.3 contains AbstFinder generated output list of abstractions.

4. Section D.4 contains the original specification file after sifting out the abstractions identifiers found by AbstFinder.

D.1 Source Transcript of the Flinger Missile Case Study

In this appendix section, long lines are folded at a convenient word break, right at the end of some word. The break is denoted by a "\" at the end of all pieces of the line but the last and at least one space at the beginning of all pieces except the first.

```
Statement of Operational Need for the
Short-Range Remote-CONTROLX Real-Time Video
Attack and Reconnaissance FLINGER-MISSILE
Draft (4/25/90)

Mission Requirements

The mission of the FLINGER-MISSILE is to provide an effective defense from
ground and air targets as well a real-time, recordable video imaging
capability.  Using real-time imaging, target acquisition and ordinance
delivery can be done remotely with very high accuracy.  Video, time, date,
location, and orientation information is collected and can be recorded at
the ground CONTROLX subsystem and later communicated to command center
locations.  Video imaging can be used to aid in the targeting capabilities
under remote CONTROLX or in autonomous mode.  The operator can CONTROLX the
FLINGER-MISSILE remotely using a joystick-type CONTROLX at the ground CONTROLX
subsystem.  In the autonomous mode, the imaging system will utilize infrared
detection methods for target selection.  CONTROLX signals going to the
FLINGER-MISSILE and information returning from the FLINGER-MISSILE will not be\
 constrained
by line-of-sight limitations.  The system should be able to operate in an
all-weather environment.

Operational Objectives

FLINGER-MISSILE Subsystem.  The FLINGER-MISSILE will have an operating range of five
```

miles when CONTROLX remotely and a range of fifty miles at cruising or reconnaissance speed and a range of twenty miles at intercept speed. The ordinance will be capable of piercing four inches of armour at intercept speed. A "loitering" navigation mode is required to allow FLINGER-MISSILE to circle a reconnaissance target in a radius of no more than 600 feet. Camera rotation will compensate for revolving image. Minimum speed for altitude sustaining cruising will be no more than 65 KNOTS and top speed at 850 KNOTS.

Ground CONTROLX Subsystem. The ground CONTROLX subsystem will allow the operator to view the real-time video signal from the FLINGER-MISSILE, guide the FLINGER-MISSILE using a wireless joystick-type interface, and mechanisms to CONTROLX the following FUNCTIONX:

1. Target Circling/Infrared Targeting/Remote CONTROLX. These three mutually exclusive FUNCTIONX are selected by the operator and will determine how the FLINGER-MISSILE will navigate. The first two are autonomous modes thereby disabling any joystick navigation or speed CONTROLX. Target Circling will make the FLINGER-MISSILE circle a designated target at cruising speed to allow for effective reconnaissance. Infrared Targeting would generally be used during the terminal stage giving navigation CONTROLX to the internal infrared sensors to engage the target at attack speed. The Remote CONTROLX mode will give complete navigation CONTROLX to the operator through use of the joystick and speed SWITCHX.

2. Speed CONTROLX. This FUNCTIONX selects between the two available speeds. The cruise speed is generally used for reconnaissance missions or when longevity of airtime is desired. The Attack speed is the top speed available.

3. Self Destruct. When this FUNCTIONX is selected, immediate destruction of the FLINGER-MISSILE using the on-board ordinance is commenced. The only exception to this is prior to launch as well as a sufficient period of time after launch to prevent 'damage' to operator. Normal operation of the Ground CONTROLX Subsystem should be continued in case of the unsuccessful completion of this FUNCTIONX.

4. Self Test. This FUNCTIONX shall be exercised immediately after the initial power on sequence and anytime after that prior to FLINGER-MISSILE launch. This test will confirm proper operation of all navigation, video, and infrared CONTROLX and communications between the FLINGER-MISSILE and ground CONTROLX SUBSYSTEM.

6. Launch FLINGER-MISSILE. The selection of this FUNCTIONX will commence launching sequence. Prior to launching, the Remote CONTROLX navigation mode shall be selected. If the system is in Self Test, the test shall immediately terminate

and launching sequence is begun.

The operator will be notified of successful completion of the actions taken by these FUNCTIONX by both audible and unique visible signals. The speed CONTROLX SWITCHX shall reside on the joystick handle.

The real-time video display will allow the operator to view the video signal from the FLINGER-MISSILE. The artificial horizon and compass graphical instruments will also be displayed on the video screen.

A low power warning signal lamp will indicate an approximate one minute of normal operation is possible on current ground CONTROLX subsystem battery. The video screen will turn off at the point when another one minute of operation without video is possible.

A low signal strength warning lamp will indicate a signal loss of 80% from the FLINGER-MISSILE. This will warn the operator of impending loss of CONTROLX.

FLINGER-MISSILE Interview - 25 April 1990

S) = Scott Stevens as the Requirements Analyst
J) = John Herman as the OPERATOR

S) Operational need, talk about it being able to cruise at 65 knots in a 60 ft circular mode. If I flip SWITCHX to go up to 850 knots and try to maintain that tight circle, something is going to happen.

J) Loitering navigation circling is going into an autonomous mode that is not CONTROLX by OPERATOR. The OPERATOR would have to move out of that using a SWITCHX to get out of autonomous mode to gain CONTROLX back.

S) Before FLINGER-MISSILE would accept the command to change speed, you then have to take it out of loitering mode into manual mode. In fact, it will not always accept all signals.

J) In the statement operational need you have two modes. Reconnaissance mode and attack mode. The second and third FUNCTIONX CONTROLX whether its in autonomous mode or remote CONTROLX mode. In the reconnaissance mode, you can operate the FLINGER-MISSILE by remote CONTROLX but when you take out of remote CONTROLX and into autonomous position, it would immediately go into loitering mode. The autonomous position on the SWITCHX is determined by what mode you are in.

S) We can go into reconnaissance (is it same as loitering).

J) No, the reconnaissance mode is separate; it affects what the other SWITCHX will do.

S) Reconnaissance mode is straight level. It does not go around in a circle.

J) Right. Reconnaissance mode has an affect on the second SWITCHX. When you move it out of remote CONTROLX and you are in the reconnaissance mode, it would go into a loiter navigation mode where it would circle.

S) Let me make sure I understand this....In reconnaissaance mode, how fast would we be going.

J) That would depend; it would actually move fairly fast. It would not affect speed if in remote CONTROLX mode.

S) Reconnaissance mode is not necessarily remote CONTROLX and it does not affect speed. What does it affect?

J) It affects the autonomous mode position.

S) If in reconnaisance and manual, I would be flying at 850 knots.

J) You could very well do that.

S) What else could I do?

J) 265 knots in reconnaissance mode.

S) The other mode is reconnaissance and loitering. I'll have one SWITCHX that SWITCHX between the two.

J) You have reconnaissance and attack mode. In attack mode, there is no loitering. For instance, the operator has the FLINGER-MISSILE, they see a target; they have two separate missions: 1) is a reconnaissance mission they want to see the troop movement on other side of bridge. They operator moves SWITCHX to mode.

S) What mode where they in?

J) It was not even on. It does not matter; it was not launched yet. This is the operator moving into position, he has a mission. His supervisor told him to go out and figure out what the troop movement is.

S) But you said he moved the SWITCHX to the reconnaissance mode. Was it off

before? One that says reconnaissance, another that says attack and another that says loiter.

J) No, they are mutally exclusive. The reconnaissance and attack mode.

S) Where does loiter come in?

J) That is a different SWITCHX.

S) So there are three SWITCHX?

J) No, there could be several SWITCHX.

S) I do remember something on the number of SWITCHX. Loiter, navigation, remote CONTROLX and autonomous. Those two share a single double-throw SWITCHX. Is that right?

J) Actually, this should be a single throw SWITCHX. It depends on the mode of the mission. Since the ground subsystem is off, the SWITCHX have to be in some position. When I said to move the SWITCHX into reconnaissance mode that is only if previously in attach mode.\ Only make sure SWITCHX is in reconnaissance mode SWITCHX. That SWITCHX being in reconnaissance mode position would affect another SWITCHX which would determine whether its remotely CONTROLX or in autonomous mode.

Because in reconnaissance mode, the FLINGER-MISSILE is fired and moved into\ position by remote CONTROLX and then the SWITCHX is moved away from the remote CONTROLX to the autonomous mode. Since it is in the reconnaissance mode, it will then go into loitering navigaton.

S) When it first takes off in reconnaissance mode manually operated, how fast is it flying?

J) It could go in either two different speeds.

S) How is that set?

J) By the speed SWITCHX.

S) We have one SWITCHX that has attack and reconnaissance and another SWITCHX that says loiter and off.

J) No, it would have either the remote CONTROLX or the autonomous.

Remote CONTROLX is the same in either reconnaissance or attack mode because you are CONTROLX the FLINGER-MISSILE by remote CONTROLX.

S) Alright, we have a speed SWITCHX. If the attack reconnaissance SWITCHX is on reconnaissance and we are in remote CONTROLX, we can set speed at 850 or 65.

If we SWITCHX to autonomous, no matter what the speed SWITCHX is set to, it drops to 65.

J) Correct, it would then circle.

S) The FLINGER-MISSILE then will not necessarily accept all of the data or would ignore some of the data. At the same time, are we getting feedback on the speed? That would not be useful to the OPERATOR?

J) The OPERATOR would be getting position information as much as orientation. No they would not. The only indication of position they would be getting would be from video.

S) The OPERATOR sees what the FLINGER-MISSILE is doing through a video monitor. We have determined it is possible to put some of the data on the monitor. You say that the subsystem receives the status from the FLINGER-MISSILE. What kind of status information is it getting back from FLINGER-MISSILE.

J) It gets orientation of the FLINGER-MISSILE...rise and climb.

S) It is getting continuous feedback on the position of things like the rudder and the elevator and so on.

J) Right, the new information is sent back only from the FLINGER-MISSILE of the orientation. The information needed for the artificial horizon and the direction.

S) So you not going to send back status information on the rudder position. If it moves, you just see that its moves. What other kind of status are we getting back.

J) We would get back on things like whether the infrared has locked onto an object. Whether the target is ready to be put into the auto mode.

S) Do you have preference for which of these items is displayed on the monitor and which items are separate lamps or some other kind of status indicator to the OPERATOR.

J) The most critical information during the remote CONTROLX mode would be position information.

S) The position information is in what form other than just seeing from the camera where you are. Anything else?

J) Right now no. Just seeting where you are on the camera determining where you would have your artificial horizon and your directional data. That right now is the minimum. We did think about getting some positioning data from the global positioning satellite. We do not know if we can do that in real time with the accurancy that we wanted, but no that is not required.

S) The specification that came out originally said that the guidance data was rudder position and elevator position. But you are saying that it is going to be direction and artificial horizon. Does that mean I am going to have to map that?

J) That is the information from the ground CONTROLX SUBSYSTEM to the\
FLINGER-MISSILE. The
FLINGER-MISSILE would then send back the artificial horizon information.

S) I do not see that anywhere. Is this a new addition?

J) This would be the statement of operational need.

S) As a OPERATOR, there is a mapping between the joystick position and the rudder position. It is not a one-to-one mapping. The effect of that is going to be at various times potentially the joystick will move different amounts of time for different rutter movements. Because you are mapping 256 positions on the joystick to 90 positions on the rutter since it is not evenly divisible, somewhere in there you are going to have one or two movements on the joystick all of a sudden mapped to one. As a OPERATOR, do you think that is going to affect anything.

J) Yes, that information was there as an example. As you saw, we did say these would not necessarily be the final values, if you want to do one-to-one mapping. That was put in before the interface data was in and that could be changed.

S) At this point, we do not really know what the values are going to be. If we get the go on this, do you think we should just start developing software so that we can handle either one-to-one mapping or other mapping, mapping it more general.

J) That would depend on your communication between the interface of the FLINGER-MISSILE software and the ground CONTROLX subsystem software. That could be whatever needs to be the information. Those were arbitrary at that point. They would be the requirments if we had the interface data.

S) Besides the gun-like SWITCHX on the joystick, what other types of SWITCHX do you think ought to be here. Are there some SWITCHX which for one reason or another ought to have either software or hardware fail safe kind of device.

J) Right, hardware fail safe device on the self-destruct.

S) Just hardware; we do not have to worry about software?

J) No, we would like to be able to self-destruct at any time.

S) You mentioned that there is an initial set of signals to be interpreted: error, fatal error, radar on, or locked. Initially means these are the ones that you've thought about so far, but there may be others?

J) Right, fatal error there is a point where it would self-destruct on its own because of an error. That would prevent the FLINGER-MISSILE from not being CONTROLX and, at that point, it would self-destruct.

S) The FLINGER-MISSILE takes care of all that. Is there some sort of identified friend or foe data in here?

J) No, that would be OPERATOR CONTROLX. The operator would have an idea. Actually, in many cases, it would not know whether it would be a friend or foe until they would get closer to the target.

D.2 Ignored Files Used in the Flinger Missile Case Study

ignored-common-word-file:

```
the of is to an a from and as can be be being do does done did with very high
later well use using used in or at on want wanted anywhere until many new
somewhere whatever will for not by should able all have has had when more than
top are how first thereby any would make made give through this here between
two when generally immediately only now sufficient after continued case cases
shall initial that thats begin gegun successful take takes taken
these both unique also
low one normal without possible ready kind most during just anything talk
about it go going went up try tight if i into move moves moved out
get gets getting like
before allways back send sent you fact second third whether its but what same
other straight where me sure how we depend actually onto affect affects
necessarily
necessary could very else instance they see side even matter told him said
off no so three several those right since some another which alright set
put much correct
becuase was there then something under even effective affect available allow
thing  either shar
throw think know knows ought different differents
really actually fairly change because things your prior selected form
loss handle seperate says determine determined prevent
```

ignored-application-words-file:

```
point knots fast single values mean minimum exclusive
850 231 range inche inches miles feet 60 65 600
flinger missile
```

ignored-suffixes-file:

```
ication cation sition tation ation ition ntion ption ction tion
ance ence ement ment sent able ered
rning ining ying ving ning ling ting ings ing
iles eeds round sist tive ious ware mage
tified pplied ified lied ited  ated ired ared ined ied ted
tially ually ally lly
```

D.3 AbstFinder Results of the Flinger Missile Case Study

```
        ************************************************
        S U M M A R Y   OF      A B S T F I N D
        ************************************************

-------------------------------------------------
# of lines read from input file is 279
# of abstractions found is 165
-------------------------------------------------
```

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
1	114	1	24	operator|
2	116	1	17	position |
3	133	1	17	position|
4	11	1	15	subsystem|
5	150	1	13	information|
6	36	1	7	launch |
7	80	1	7	launch|
8	111	1	6	circle|
9	63	1	6	circle|
10	126	1	5	status |
11	141	1	5	mapping|
12	146	1	5	mapping|
13	121	1	4	orientation |
14	82	1	4	troop movement |
15	118	1	3	monitor|
16	33	1	2	self test|
17	103	1	2	speeds|
18	85	1	2	mutually |
19	106	2	42	remote controlx |remote controlx autonomous|
20	104	2	37	speed |speed switchx|
21	73	2	28	switchx|x switchx |
22	88	2	28	switchx|j switchx|
23	89	2	28	switchx|s switchx|
24	90	2	28	switchx|j switchx|
25	91	2	28	switchx|r switchx|
26	93	2	28	switchx|e switchx|
27	94	2	28	switchx|j switchx|
28	144	2	25	operator |2 oper|
29	163	2	24	operator |operator i|
30	138	2	16	information|artificial horizon |

31	16	2	15	cruising |speed |
32	87	2	13	loiter|loiter |
33	145	2	13	information |s information |
34	79	2	12	s mode|s mode |
35	10	2	11	operational |2 oper|
36	164	2	11	r target|friend foe |
37	139	2	10	statement operational need|j statement operational need|
38	55	2	10	statement operational need |j statement operational need|
39	128	2	10	target|target a|
40	1	2	9	mission |requirements|

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
41	157	2	9	signals in|s inter|
42	148	2	8	mapping |software |
43	154	2	8	self destruct|hardware fail safe device|
44	29	2	6	destru|self destruct|
45	156	2	6	self destruct|self destruct |
46	112	2	6	s accept |e data|
47	155	2	5	software|hardware |
48	113	2	5	ack speed|feedback |
49	15	2	4	ate re|camera |
50	162	3	53	controlx|operator controlx|j operator |
51	25	3	45	controlx|speed |speed controlx|
52	17	3	44	controlx |ground |ground controlx subsystem|
53	160	3	42	controlx |controlx s|self destruct|
54	108	3	37	speed |speed switchx|s speed switchx|
55	71	3	31	reconnaissance |reconnaissance m|reconnaissance mode|
56	119	3	19	subsystem |status |s status |
57	54	3	12	signals|cept s|s accept |
58	26	3	10	functionx|functionx |speeds|
59	136	3	7	ection |artificial horizon|direction|
60	134	3	7	real time |accuracy|required|
61	127	3	7	infrared |9 j inf|locked|
62	13	3	5	ordinance |intercept speed|ordinance c|
63	161	3	5	e data|e data |friend foe |
64	158	3	3	locked|error |fatal error |
65	47	4	53	controlx|operator |n operator|operator i|
66	67	4	48	remote controlx |speed |speed re|remote controlx mode|
67	38	4	46	remote controlx |navigation mode |launch|launching |
68	84	4	38	attack |reconnaissance |loiter|reconnaissance attack |
69	9	4	36	system |operat|system operat|operate |
70	60	4	36	reconnaissance |loitering |s reconnaissance |reconnaissance loiteri
71	50	4	33	circle |switchx|switchx |switchx m|
72	66	4	31	reconnaissance |reconnaissance m|reconnaissance mode |7 reconnaissa
73	31	4	30	operator|launch|time launch|launch |
74	48	4	26	requirements|operator|n operator|requirements |
75	124	4	18	information |ection|artificial horizon |artificial horizon directio
76	120	4	16	information |s information |status |status information |
77	49	4	15	operational need |t circ|cruise |s oper|
78	123	4	14	orientation|information |information orientation|j information |
79	159	4	7	self destruct|self destruct |error |fatal error |
80	147	4	6	interface |e data|interface data|e data |

#)	abst#	corr_ phras#	corr_ lines#	correlateted-phrases
81	81	5	42	mission|operator |position |e mission|operator m|
82	77	5	39	reconnaissance |mission |reconnaissance mission|1 reconnaissance m|
83	152	5	34	joystick type|switchx|switchx |s switchx|s switchx |
84	86	5	34	attack |reconnaissance |attack mode|7 reconnaissance |reconnaissanc
85	153	5	32	switchx|switchx re|57 switchx re|software |hardware fail safe devic
86	69	5	30	autonomous mode|autonomous mode |position|mode p|mode position|
87	131	5	28	information |camera |s position |position information |seeing camer
88	28	5	25	attack |speed |attack speed|ck speed s|d speed |
89	95	5	20	mission|s mode|s mode |mode m|e mission|
90	142	5	12	time r|joystick |movement|movements|rutter |
91	132	5	9	ection|camera |artificial horizon |artificial horizon direction|see
92	21	6	57	controlx|autonomous mode|joystick |speed |navigation |speed control
93	137	6	54	controlx |ground |information |ground controlx subsystem |j informa
94	92	6	52	remote controlx |remote controlx autonomous|navigation |loiter|r na
95	110	6	45	x autonomous |speed |speed switchx|switchx |switchx autonomous |s s
96	56	6	34	attack |reconnaissance |reconnaissance m|reconnaissance mode |attac
97	74	6	34	attack |reconnaissance |attack mode|j reconnaissance |reconnaissanc
98	117	6	33	video |operator |s oper|operator s|monitor|s operator |
99	70	6	32	reconnaissance |reconnaissance m|manual |s reconnaissance |8 s reco
100	45	6	18	video |operation|video s|operation |video screen|minute operation |
101	135	6	17	position |r position |ator position|rudder |elevator |rudder positi
102	30	6	16	ordinance |ordinance c|functionx|functionx |destru|commence|
103	46	6	10	signal|signal |nal st|e signal|warning |lamp indicate |
104	151	6	9	requirements|interface |requirements |e data|interface data|s inter
105	96	7	56	ground |subsystem |switchx|system s|switchx |position|subsystem s|
106	78	7	52	operator |switchx|x mode|switchx |switchx m|3 operator switchx |swi
107	64	7	50	reconnaissance |switchx|reconnaissance m|reconnaissance mode |recon
108	19	7	47	remote controlx|target|target |targeting |infrared |target circling
109	101	7	41	reconnaissance |loitering navigation|reconnaissance m|reconnaissanc
110	42	7	38	video |real time video |operator |signal|operator view |video signa
111	20	7	33	operator |functionx|x operator |functionx |operator n|r navig|mutua
112	62	7	32	reconnaissance |reconnaissance m|reconnaissance mode |s reconnaissa
113	75	7	22	attack |loitering|attack mode|mode l|mode loiter|mode loitering|att
114	122	7	19	s cont|position |position r|feedback |rudder |elevator |ck position
115	43	7	16	video |video s|display|video screen|artificial horizon |s displayed
116	34	7	15	functionx|functionx |launch|time launch|sequence|sequence |power |
117	22	8	55	reconnaissance|target|target |cruising |speed |circle |target circl
118	83	8	50	reconnaissance |switchx|reconnaissance m|switchx re|reconnaissance
119	59	8	49	autonomous |switchx|x mode |switchx |switchx m|autonomous position
120	39	8	24	system |termina|e launch|self test|sequence |launching sequence|sys

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
121	12	9	68	controlx re|reconnaissance |remote|remotely |controlx remotely |int
122	8	9	51	controlx |information |controlx s|signal|signals|s information |s i
123	149	9	49	controlx |ground |ground controlx subsystem |commun|interface |comm
124	27	9	49	reconnaissance |mission|time d|speed |speed reconnaissance|cruise
125	72	9	45	reconnaissance |e reco|s mode|s mode |mode re|e reconnaissance |loi
126	102	9	41	reconnaissance |operat|reconnaissance m|mode man|reconnaissance moc
127	40	9	36	operator |signals|functionx|functionx |operator n|comple|completion
128	53	9	33	command |speed |loitering |d speed |s accept |loitering mode|manual
129	143	9	30	joystick |s joystick |joystick s|visible |position|movement|mapping
130	125	9	29	information |s information r|position|r position|s status |status i
131	52	10	79	controlx |x autonomous mode|operator |autonomous mode |switchx|swit
132	41	10	67	controlx |controlx s|joystick |speed |speed controlx|switchx|switch
133	130	10	60	remote controlx |information |information re|al inf|remote controlx
134	32	10	57	operation|controlx |ground |ground controlx subsystem |system u|fun
135	105	10	57	attack reconnaissance |reconnaissance s|loiter|switchx|switchx |swi
136	61	10	52	reconnaissance |switchx|reconnaissance m|switchx |reconnaissance mo
137	97	10	52	reconnaissance |switchx|reconnaissance m|switchx re|reconnaissance
138	5	10	50	video |remote controlx |capabilit|video imaging |autonomous mode|ta
139	76	10	40	mission|r target|target |operator |target se|missions|1 oper|ate mi
140	51	11	69	e controlx |autonomous mode|operator|autonomous mode |loitering nav
141	6	11	63	remote|controlx re|e controlx |ground |remotely |ground controlx su
142	107	11	62	remote controlx re|attack |reconnaissance |controlx remote|mode co
143	14	11	53	reconnaissance |e reco|target|target |navigation |circle |navigatio
144	98	11	50	reconnaissance |switchx|reconnaissance m|switchx re|reconnaissance
145	115	11	47	information |orientation|operator |position |information orientatio
146	129	11	35	operator|r oper|ate la|display|s displayed |indicat|separate |monit
147	3	11	33	real time |remote|target|imaging |remotely |e imaging |target |ordi
148	37	11	19	ection |selection|functionx|functionx |commence|launch|sequence|x c
149	109	12	80	reconnaissance |attack reconnaissance |remote controlx |controlx s|
150	68	12	60	remote controlx |reconnaissance |controlx s|mode re|reconnaissance
151	44	12	58	controlx |operation|ground |ground controlx subsystem |signal|signa
152	7	12	45	imaging |target|e imaging |target |system |autonomous mode|infrared
153	2	12	34	real time |video |mission |target|imaging |record|ground |capabilit
154	100	13	90	reconnaissance |remote controlx |y remote|controlx s|remote control
155	4	13	60	controlx |video |record|ground |ground controlx subsystem|system |i
156	35	13	58	operation|controlx |video |ground |video i|commun|ground controlx s
157	57	13	53	controlx |remote controlx |controlx autonomous mode|autonomous mode
158	140	13	45	joystick |operator |s oper|position|position |operator m|position r
159	18	14	76	controlx |real time video |ground |ground controlx subsystem |opera
160	0	14	72	real time |video |remote|controlx |remote controlx |controlx re|ope
161	23	14	57	controlx |attack |target|target a|targeting |infrared |navigation

```
162    | 58    | 16    | 93    | remote controlx re|reconnaissance |remote controlx autonomous |oper
163    | 65    | 17    | 67    | remote controlx re|reconnaissance |loiter|navigation mode |circle|r
164    | 99    | 18    | 90    | reconnaissance |controlx |remote|remotely |controlx autonomous mode
```

D.4 The Flinger Missile Transcript after Straining

```
 of   for the short        and    draft  4 25 90

the  of the   is to provide an effective defense from  and air targets as well\
     recordable   capability
  using      acquisition and  delivery can be done  with very high
      date  location  and   is collected and can be recorded at the    and later\
 communicated to  center locations
    can be used to aid  the  capabilities under   or
  the  can  the    using     at the
   the    the   will utilize  detection methods for
    going to the   and  returning from the   will not be constrained by line of sight\
 limitations
  the  should be able to   an all weather environment

 objectives

  the   will have an operating  of five  when   and   of fifty  at  or   and   of\
 twenty  at
  the  will be capable of piercing four  of armour at
        is  to allow   to       radius of no more than
   rotation will compensate for revolving image
    for altitude sustaining  will be no more than   and top  at

  the    will allow the  to  the     from the    guide the    using  wireless     and\
 mechanisms to   the following

1

  these three     are selected by the  and will determine how the    will navigate
  the first two are  modes thereby disabling  any   or
    will make the      designated  at   to allow for effective
    would generally be used during the  terminal stage giving   to the internal\
  sensors to  engage the  at
  the    will give complete   to the  through use of the  and

  this  selects between the two available
   the   is generally used for   or when  longevity of airtime is desired
  the   is the top  available

3

  when this  is selected  immediate destruction of  the   using the on board  is\
 commenced
```

the only exception to this is prior to as well as sufficient period of after\
to prevent damage to
normal of the should be continued case of the unsuccessful of this

4

this shall be exercised immediately after the initial on and anytime after that\
prior to
this will confirm proper of all and and communications between the and

6

the of this will
prior to the shall be selected
if the is the shall immediately terminate and is begun

the will be notified of successful of the actions taken by these by both audible\
and unique
the shall reside on the handle

the will allow the to the from the
the and compass graphical instruments will also be on the

low will an approximate one of normal is possible on current battery
the will turn off at the when another one of without is possible

low strength will loss of 80 from the
this will warn the of impending loss of

interview 25 april 1990
scott stevens as the analyst john herman as the
talk about it being able to at ft circular
if flip to go up to and try to maintain that tight something is going to\
happen

is going into an that is not by
the would have to move out of that using to get out of to gain back

before would the to change you then have to take it out of into
fact it will not always all

the you have two modes
and
the second and third whether its or
the you can the by but when you take out of and into it would\

immediately go into
the on the is determined by what you are

we can go into is it same as

no the is it affects what the other will do

is straight level
it does not go around

right
has an affect on the second
when you move it out of and you are the it would go into where it\
would

let me make sure understand this

how would we be going

that would depend it would actually move fairly
it would not affect if

is not necessarily and it does not affect
what does it affect

it affects the

if and would be at

you could very well do that

what else could do

265

the other is and
ll have one that between the two

you have and
there is no
for instance the has the they see they have two
1 is they want to see the on other side of bridge
they moves to

what where they

it was not even on
it does not matter it was not launched yet
this is the moving into   he has
his supervisor told him to go out and figure out what the   is

but you said he moved the to the
was it off   before
one that says   another that says and another that says

no they are
the and

where does come

that is different

so there are three

no there could be several

do remember something on the number of
and
those two share   double throw
is that right

actually this should be   throw
it depends on the   of the
since the   is off   the have to be some
when said to move the   into   that is only if previously attach
only make sure is
that being   would affect another which would determine whether its   or
because   the   is fired and moved into by   and then the is moved away from\
the   to the
since it is the   it will then go into

when it first takes off   manually operated how is   it

it could go either two different

how is that set

by the

we have one that has and and another that says and off

no it would have either the or the
is the same either or because you are the by

alright we have
if the is on and we are we can set at or

if we to no matter what the is set to it drops to

correct it would then

the then will not necessarily all of the or would ignore some of the
at the same are we getting on the
that would not be useful to the

the would be getting as much as
no they would not
the only indication of they would be getting would be from

the sees what the is doing through
we have determined it is possible to put some of the on the
you say that the receives the from the
what kind of is it getting back from

it gets of the

rise and climb

it is getting continuous on the of things like the and the and so on

right the new is sent back only from the of the
the needed for the and the

so you not going to send back on the
if it moves you just see that its moves
what other kind of are we getting back

we would get back on things like whether the has onto an object
whether the is ready to be put into the auto

do you have preference for which of these items is on the and which items are\
lamps or some other kind of indicator to the

the most critical during the would be

the is what form other than just from the where you are

anything else

right now no
just where you are on the determining where you would have your and your\
directional
that right now is the
we did think about getting some positioning from the global positioning satellite
we do not know if we can do that with the that we wanted but no that is not

the specification that came out originally said that the guidance was and
but you are saying that it is going to be and
does that am going to have to map that

that is the from the to the
the would then send back the

do not see that anywhere
is this new addition

this would be the of

as there is between the and the
it is not one to one
the effect of that is going to be at various times potentially the will move\
different amounts of for different
because you are 256 positions on the to 90 positions on the since it is not\
evenly divisible somewhere there you are going to have one or two on the all\
of sudden mapped to one
as do you think that is going to affect anything

yes that was there as an example
as you saw we did say these would not necessarily be the final if you want to\
do one to one
that was put before the was and that could be changed

at this we do not really know what the are going to be
if we get the go on this do you think we should just start developing so that we\
can handle either one to one or other it more general

that would depend on your between the of the and the
that could be whatever needs to be the
those were arbitrary at that
they would be the if we had the

besides the gun like on the what other types of do you think ought to be here

are there some which for one reason or another ought to have either or kind\
of

right on the

just we do not have to worry about

no we would like to be able to at any

you mentioned that there is an initial set of to be interpreted
    radar on or
 initially means these are the ones that you ve thought about so far but there\
may be others

right there is where it would on its own because of an
that would prevent the from not being and at that it would

the takes care of all that
is there some sort of identified or here

no that would be
the would have an idea
actually many cases it would not know whether it would be or until they\
would get closer to the

Appendix E Results and Data of the RFP Case Study

This appendix contains AbstFinder data and results of the RFP case study:

1. Section E.1 contains part of the RFP transcript that was used as the specification document in this case study.

2. Section E.2 contains the ignored files that were used by AbstFinder.

3. Sections E.3 and E.4 contain AbstFinder generated output list of abstractions from first and last iteration.

4. Section E.5 contains part of the already human made results.

E.1 Source Transcript of the RFP Case Study

Below are the first few pages of the RFP to allow the reader to get a sense of what is there.

```
                    SYSTEM SPECIFICATION
                            FOR
            UNMANNED AERIAL VEHICLE-SHORT RANGE
                          (UAV-SR)
                           SYSTEM


1.  SCOPE

1.1  This specification establishes the performance, design, development,
test, and production requirements for the Unmanned Aerial Vehicle-
Short Range (UAV-SR) System, hereinafter referred to as the UAV-SR System.

2.  APPLICABLE DOCUMENTS

2.1  Government Documents.  The following documents, of the issue in
effect on the date of Issuance of Request for Proposal by the Government,
unless otherwise specified, form a part of this specification to the extent
specified herein. In the event of conflict between the documents referenced
herein and the contents of this specification, the contents of this
specification shall be considered a superseding requirement.  The
contractual applicability of the references herein are according to the
category stated in the document listing below.

2.2  Document Tailoring.  These documents have been tailored by the
Government to the maximum extent possible.  The applicable portion of each
document is stated in the text of the specification where the document is
```

called out. If no portion is stated the entire document applies.

SPECIFICATIONS

(A long list of other documents is omitted)

3. REQUIREMENTS

3.1 System definition. The UAV-SR System shall provide the Military services with a delivered system, defined as all hardware and software necessary to meet the requirements of this section.

3.1.1 General description.

3.1.1.1 System Utilization. The system will initially be utilized by the U.S. Army in support of Corp Commanders and the USMC in support of Brigade Commanders.

3.1.1.2 System Configuration.

3.1.1.2.1 Description and Purpose. This section establishes the requirement for a UAV-SR System and optional mission hardware. The system consists of air vehicles (AVs), Modular Mission Payloads (MMPs), Airborne Data Terminal (ADT), Ground Data Terminal (GDT), Mission Planning Control Station (MPCS), Launch/Recovery (L/R) equipment, support equipment, and external interface hardware.

3.1.1.2.2 UAV-SR System Configuration.

The UAV-SR systems shall be delivered to the Government and be configured to meet the specific operational needs of each gaining service. See Table 3.1

TABLE 3.1

SYSTEM BASELINE CONFIGURATION

System Component	QUANTITY
Mission Planning Control Station (MPCS)	1
Mission Planning Station (MPS)	1
Ground Control Station (GCS)	2
Ground Data Terminal (GDT)	2
Remote Video Terminal (RVT)	4
Launch and Recovery (L/R) Equipment	1
Air Vehicle (AV) w/Airborne Data Terminal (ADT)	8

Ground Support Equipment (GSE)	1
Modular Mission Payload (MMP)	
Day/Night IMINT (D/N) MMP	8
Day Only (DAY) MMP	4
Airborne Data Relay (ADR) Payload	4

3.1.1.2.3 UAV-SR Equipment. The UAV-SR System shall include the following equipment:

3.1.1.2.3.1 Air Vehicle (AV). The air vehicle, (comprised of an airframe, power plant, guidance and control equipment, position/navigation equipment, removable on board storage recorder, FAA/IFF transponder and navigation lights, removeable emergency recovery system and airborne data terminal (ADT)), is the airborne component of the system whose mission is to serve as the "carrying device" for mission payloads or data link relay.

3.1.1.2.3.2 Airborne Data Terminal Equipment (ADT). This is the airborne portion of the data link equipment used for the transmission of sensor and AV status data from the AV to the GDT or the Airborne Data Relay (ADR) and the reception of command data from the GDT or the ADR. The data link(s) provide a near-real-time flow of data (sufficient to provide for effective, timely target acquisition and command and control) to and from the UAV-SR.

3.1.1.2.3.3 Mission Planning and Control Station (MPCS). The MPCS is the operational control center of the UAV-SR System and shall consist of three shelters and installed operating equipment each mounted on a vehicle, two Ground Data Terminals (GDT), and four Remote Video Terminals (RVT). One of the shelters will serve as the Mission Planning Station (MPS) and the others will serve as Ground Control Stations (GCS). Contained within the appropriate shelters are ground-based navigation equipment; launch, guidance, and recovery control equipment; data processing equipment; communication equipment; power supplies and the data interfaces to other systems.

3.1.1.2.3.3.1 Mission Planning Station (MPS). The MPS shall provide facilities to plan flights for the performance of assigned missions and will serve as a command post for the mission commander and communication equipment for the receipt of mission assignments from supported headquarters; for reporting of acquired data to the supported operations centers, e.g., AFATDS/TACFIRE, to enable timely engagements of targets located by the UAV-SR system.

3.1.1.2.3.3.2 Ground Control Station (GCS). The GCS shall provide facilities for Air Vehicle (AV) control (launch, in flight, and recovery),

GDT control, relay control, mission payload control, imagery display, receiving and transmitting data from the AV and to other MPCS units.

3.1.1.2.3.3.3 Ground Data Terminal (GDT). The GDT is the ground based portion of the data link. It consists of a controllable antenna which shall be remoted a minimum of 400 meters from the GCS via redundant fiber optic cables, a Data Link Interface Unit (DLIU) which connects to the GCS bus structure, and the necessary power supply and distribution equipment. The GDT receives and transmits data to the AV and is controlled by the GCS. One GDT is attached to each GCS.

3.1.1.2.3.3.4 Remote Video Terminals (RVT). The RVT will be provided power by the host system. See Appendix 100.

3.1.1.2.3.3.4.1 The RVT equipment shall receive, display, record, playback and freeze IMINT sensor data (overlayed with AV position and heading, North seeking arrow, time, and target coordinates) at a location other than a control station (MPCS or L/R).

3.1.1.2.3.3.5 Ground-Based Support Navigation Equipment. This includes equipment which may be located within the MPCS that is associated with the processing of AV location and other data required for meeting mission objectives.

3.1.1.2.3.3.6 Guidance and Control Equipment. This includes guidance and control equipment which may be located within the MPCS as well as the ground-based portion of the data link(s) and any guidance controls and display consoles, command/video/telemetry instrumentation and recording capabilities associated with performance of required mission objectives.

3.1.1.2.3.3.7 Data Processing Equipment. The MPCS also records and plays back imagery data collected from all on-board modular mission payloads (MMP) and, based upon the nature of the collected data, transmits selected data in standard message text format such as RECCEXREPS and TACREPS, to the supported unit. The MPCS is desired to be highly automated with aids to assist operators in mission planning, MMP processing (which consists of the ability to edit and condense the tape material, freeze frame, digitize and produce a hard copy of an image), and report dissemination (standard message text format for transmission on communications equipment or external interface ports). This includes any computer and signal processing equipment.

3.1.1.2.3.3.8 Communication Equipment. This is currently fielded communication equipment which shall be used for communicating with supported units, support of command and control, and report dissemination.

3.1.1.2.3.3.9 External Interfaces. All external interfaces shall be conducted through standard format ports (eg; RS232, RS170) on the MPS and GCS units. These ports shall be capable of transmitting video and digital data.

3.1.1.2.3.4 Launch and Recovery. Equipment utilized to launch and recover the AV. This equipment will also support BIT/BITE on the AV.

3.1.1.2.3.4.1 Launch System (LS). Equipment required to launch the air vehicle with modular mission payloads into flight, check out AV and payload, and hand-off to a GCS.

3.1.1.2.3.4.2 Recovery System (RS). Equipment required to recover the air vehicle with modular mission payloads after mission completion or upon command.

3.1.1.2.3.5 Modular Mission Payloads (MMP). Modular mission payloads are defined as modular sensor equipment installed on-board the air vehicle to accomplish the mission with the exception of equipment used to launch, fly, navigate, and recover the air vehicle. Data link(s) and modular on-board data storage recorder equipment are not considered a part of the MMP.

3.1.1.2.3.5.1 Day/Night Passive IMINT MMP. Sensor equipment installed on-board the AV which shall provide near-real-time imagery of the target area both day and night.

3.1.1.2.3.5.2 Day Only Passive IMINT MMP. Sensor equipment installed on-board the AV which shall provide near-real-time imagery of the target area during the day.

3.1.1.2.3.6 Airborne Data Relay (ADR) Payload. Equipment installed on board the AV to accomplish two-way data link relay capability between the ground and another AV carrying a sensor MMP.

3.1.1.2.3.7 Equipment Interface Hardware. Hardware, provided by the contractor, necessary to interface the various components of the UAV-SR System including, but not limited to, breakout boxes, ATE adapters, cables, RF Test Boxes, etc., as required for operation, maintenance, and training.

3.1.1.2.3.8 Support Equipment. The UAV-SR support equipment includes all operating, transportation, training, handling, maintenance, and checkout equipment. This equipment is desired to be held to the lowest possible

density and complexity necessary to satisfy mission and force structure requirements to include the proposed training devices by the contractor. Military standard test equipment shall be used to the maximum extent possible.

3.1.1.2.3.8.1 Operating and Handling Equipment. The operating and handling equipment includes all special equipment, shelters, vehicles, integrated power environmental control systems, slings, wiring harnesses, power generators and all other equipment necessary to operate, support, and transport (air, sea, rail, truck, or helicopter lift) the UAV-SR System. Standard Government shelters or Light Weight Rigid Wall Tactical Shelter (compatible with HMMWV model 10001) are required to be used for production. The production UAV-SR System is required to be capable of operation and transport on standard High Mobility Multipurpose Wheeled Vehicles (HMMWV), 5-ton trucks and trailers (See paragraph 3.2.8).

3.1.1.2.3.8.2 Maintenance Equipment. Equipment used to maintain the UAV-SR System. This includes all equipment used to maintain, calibrate, and repair the system. It also includes all Test, Measurement, and Diagnostic Equipment (TMDE).

3.1.1.2.3.9 Modular On-board Storage Recorder. Equipment used to record and playback payload and AV data during autonomous operations, in the event of loss of data link, and on operator command.

3.1.1.2.4 UAV-SR Functional Areas. (See Appendix 10)

3.1.2 Missions. (See Appendix 200)

3.1.3 Threat. (See Appendix 200)

3.1.4 System Diagram. (See figure 1)

3.1.5 Interface Definition. The GCS and MPS shall be capable of providing communications with TACFIRE/AFATDS and ASAS. The capability to automatically log all incoming and outgoing formatted tactical messages shall be provided. All digital messages received from TACFIRE/AFATDS and ASAS shall be error checked automatically. TACFIRE/AFATDS and ASAS messages that contain errors which are detected but cannot be corrected automatically shall be discarded. The system shall also have the ability to free form text messages in formats TBD by the operator on a word processor.

E.2 Ignored Files used in the RFP Case Study

ignored-common-words-file:

```
somewhere whatever will for not by should able having have has had
top are how first thereby any would make made give given throughout through
this adhere herein here between upon their greater better full
two when generally immediately only now sufficient after continued case cases
shall initial that thats begin begun successful take takes taken
these both unique also toward further neither
low one four  fourth five
normal within without possible already ready kind least most during
just anything talk about it go going went down up try tight if i into
move moves moved  outside out may
get gets getting like
before allways back send sent you fact second third whether its but what same
other straight anywhere me sure how we depend actually onto affect affects
necessarily
necessary could very else instance they see side even matter told him said
off no non so three several those right since some another which alright set
put much correct
was there then something under even effective affect available
thing  either each shar stand consider considered describe described
throw think know knows ought different differents
really actually fairly change because things your prior selected form
overall allow all via such over
loss handle seperate says determine determined prevent
provide provides provided show showed shown
included includes including include
utilize utilized existing follow follows following change changes
current near above below bottom per unless less otherwise
the of is to an a from and as cannot can been be being do does done did
with very high where until many new while when more than among
later well use using used in or at on want wanted
```

ignored-application-words-file:

```
air vehicles vehicle uav av sr manned
operational operations operation operating handling handled examples example
requirement requirements require required requiring engineering
specification specifications specified  short range systems system
knots meters inches ton american society arizona
computer software hardware mil dod std cat ieee
table figures figure diagram curve lines line margin format type
```

```
note appendix article paragraph paragraphs para section sections
listing contents chapter chapters series
definition defined define apply applied based denotes
consists consist contained containing contain meet increase
delete deleted dated subjected continuous continue extent maximum minimum min
ability general description purpose due needs need  plus
46168 11991
23377 83286 10304 5044 5400 7793e 2167a 2168 1472 7075 8010 7438 july 1989
1000 810d 110of 160of 128k
510 280 000 462 514 125 400 232 100 200 454 220 110 117 516 779
46  45 87 14 15 01 02 03 04 05 06 10 11 12 20 35 40 50 60  70 31
0 1 2 3 4 5 6 7 8 9
iii faa iff iia iaw ii
a b c d e f g h i
ce cs rs kt hr mhz  hz cm meter km db feet kb volt ber fahrenheit u s army
```

ignored-suffixes-file:

```
lications tification ification ilization ization
ication cation stration tration eration ibration ration sition ditions dition
mentation itation tation iation tration ulation llation lation
igation minationination uation dation ation
ition ention ntion ception ption olution
tection jection lection ection uction ction tion ension nsion usion sion
formance enance dance urance arance rance tance ance ference ence encies
rement agement gement mplement lement ovement ement nment ishment hment ment
sent cident fficient ent tible ssible ible
sable eable table inable nable iable hable able
aining rning ering ining ying ving ning ling icating ulating lating ating
ifting sting itting tting eting uting cting ting
nding iding ading ding ring ings
essing sing pping icing cing lizing ishing
iles ile eeds round sist ious ware mage age
tified pplied ified lied ited tected icted rected ected cted rated
ulated ulate lated lat ints vices ories
ated mitted dered ired ire ered ared are fined tained ined ied ted ished
tially ually ically ally lly alled ded ized imize ceiver
ability sibility ibility eability bility lity ctivity ilities istics
ective ctive  ative osive tive tical nical ical
ature ure ternal minal inal uffer plays play ology
lters ters 00 esign ign plicate icate ate ainer ntain
ine atic tic ote ter le ight ght accord titude cular cators ctors itors tors tural
945 01 05 20 30 87 37 40 46 47 49 14 15 16 25 60
0 1 2 3 4 5 6 7 8 9
ive er ed ing out rtial ial ity
```

E.3 AbstFinder Results on the RFP Case Study: First Iteration

```
        ***************************************************
        S U M M A R Y   OF      A B S T F I N D
        ***************************************************

-------------------------------------------------------
# of lines read from input file is 1920
# of abstractions found is 1627
-------------------------------------------------------

------  |------ |------ |------ |--------------------
 #)     | abst# | corr_ | corr_ | correlateted-phrases
        |       |phras# |lines# |
------  |------ |------ |------ |--------------------
 1      | 12    | 1     | 226   |  equipment|
 2      | 13    | 1     | 226   |  equipment|
 3      | 72    | 1     | 226   |  equipment|
 4      | 503   | 1     | 226   |  equipment|
 5      | 647   | 1     | 226   |  equipment|
 6      | 1002  | 1     | 226   |  equipment|
 7      | 1181  | 1     | 142   |  mission |
 8      | 1055  | 1     | 113   |  test |
 9      | 85    | 1     | 95    |  interface |
 10     | 241   | 1     | 87    |  transponder|
 11     | 1406  | 1     | 87    |  transponder |
 12     | 474   | 1     | 82    |  launch recovery l r |
 13     | 476   | 1     | 82    |  launch recovery|
 14     | 927   | 1     | 82    |  launch recovery l r |
 15     | 50    | 1     | 82    |  launch recovery|
 16     | 199   | 1     | 66    |  performance|
 17     | 55    | 1     | 59    |  recovery |
 18     | 1599  | 1     | 59    |  recovery |
 19     | 1193  | 1     | 57    |  launch|
 20     | 3     | 1     | 40    |  configuration|
 21     | 6     | 1     | 40    |  configuration|
 22     | 472   | 1     | 37    |  remote video terminal rvt |
 23     | 335   | 1     | 30    |  link |
 24     | 145   | 1     | 30    |  communication |
 25     | 860   | 1     | 29    |  training|
 26     | 922   | 1     | 29    |  shelters|
 27     | 1263  | 1     | 27    |  temperature|
 28     | 1267  | 1     | 27    |  temperature|
 29     | 858   | 1     | 25    |  personnel|
 30     | 519   | 1     | 23    |  growth |
```

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
31	880	1	23	characteristics
32	112	1	23	characteristics
33	1401	1	23	growth
34	635	1	22	altitude
35	223	1	22	altitude
36	677	1	22	altitude
37	1260	1	22	altitude
38	228	1	19	speed
39	645	1	18	vibration
40	489	1	18	generators
41	1291	1	18	vibration
42	1370	1	18	vibration
43	171	1	18	generators
44	882	1	17	shelter
45	1598	1	17	action
46	1035	1	16	ilsds
47	1082	1	16	analysis
48	803	1	15	safety
49	809	1	15	safety
50	1525	1	15	field
51	1535	1	15	position
52	671	1	14	radiation
53	564	1	14	criteria
54	1582	1	14	real time
55	777	1	13	finish
56	105	1	13	organization
57	1434	1	13	mtbf
58	620	1	12	fungus
59	1276	1	12	fungus
60	1359	1	12	fungus

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
61	1576	1	12	navigation
62	589	1	11	applicable
63	1379	1	11	exceed
64	681	1	11	rain
65	603	1	10	humidity
66	1178	1	10	profile
67	1271	1	10	humidity
68	1306	1	10	load p
69	1349	1	10	humidity
70	1441	1	10	mtbmcf
71	1464	1	10	mtbmcf
72	1469	1	10	mtbmcf
73	1403	1	9	30 analog bandwidth
74	947	1	8	instrumentation
75	970	1	8	grounding terminal
76	185	1	8	survivability
77	1221	1	8	ilsds
78	1353	1	8	dust
79	1363	1	8	acceleration
80	1418	1	8	fuel
81	656	1	8	mechanical shock
82	664	1	8	acceleration
83	84	1	7	threat
84	1433	1	7	definitions
85	1611	1	7	telemetry
86	1154	1	6	retest
87	525	1	6	elapse time
88	221	1	6	endurance
89	1281	1	6	icing
90	1284	1	6	salt fog

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
91	1355	1	6	salt fog
92	543	1	6	21 main memory
93	613	1	6	icing
94	1555	1	6	lat long
95	626	1	6	salt fog
96	580	1	6	availability
97	1301	1	5	loaded
98	523	1	5	coverage
99	998	1	5	lighting
100	904	1	5	filters
101	902	1	5	filters
102	1491	1	5	bit error rate
103	1340	1	4	sand dust
104	623	1	4	color
105	1136	1	4	burn test
106	226	1	4	maneuver
107	554	1	4	water
108	906	1	4	exhaust blower
109	650	1	4	exception
110	1563	1	4	meteorolog
111	648	1	4	exception
112	1195	1	3	penetration
113	91	1	3	legend
114	1038	1	3	regulations
115	837	1	3	plastic
116	890	1	3	inter
117	622	1	3	s fung
118	1	1	3	utilization
119	796	1	3	workmanship
120	854	1	3	logistics

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
151	28	2	105	data terminal | ground data terminal gdt
152	132	2	105	data terminal | ground data terminal gdt
153	1479	2	92	ground |l ground
154	643	2	89	environment| induced environment|
155	1243	2	87	trans| transit
156	910	2	80	electr| electrical power|
157	848	2	79	electron| r electr|
158	1122	2	69	performance | performance parameters|
159	1339	2	69	mpcs l| mpcs l r
160	1473	2	66	system| subsystem
161	215	2	61	recovery | emergency recovery
162	542	2	60	capability |expansion c|
163	140	2	59	recovery | recovery site|
164	53	2	57	launch | launch ls
165	139	2	57	launch | launch site|
166	1560	2	57	s launch | ls launch
167	364	2	54	airborne data | airborne data relay adr
168	9	2	52	component |quantity|
169	1489	2	42	operator| avo operator|
170	8	2	40	configuration| baseline configuration|
171	997	2	39	mination| shelter
172	442	2	38	display | display s|
173	676	2	38	condition| condition
174	33	2	37	remote video terminal| remote video terminals rvt
175	1600	2	37	t remote video terminal| rvt remote video terminal|
176	1124	2	36	functional | functional audit|
177	62	2	35	imint |passive imint mmp|
178	1621	2	35	station| ws work station|
179	1241	2	33	tactical | vibration|
180	870	2	31	relay| relays|

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
181	887	2	30	temperature |limits|
182	493	2	30	communication| communications|
183	1219	2	30	training| surrogate training|
184	1248	2	30	testing| e3 te|
185	1448	2	30	operate | operate time|
186	124	2	30	video | video bus|
187	1129	2	28	accepta| acceptance tests|
188	1168	2	26	reliab| reliability|
189	556	2	26	reliab| reliability|
190	888	2	25	personnel | personnel space|
191	1591	2	24	position navigation| posnav |
192	349	2	23	digital| cmd digital|
193	348	2	23	digital| cmd digital|
194	1400	2	23	growth goal| growth goals |
195	772	2	22	report |d usamicom technical report rd te |
196	770	2	22	report | usamicom technical report rd te |
197	988	2	21	phase l|e load|
198	246	2	21	navigation | navigation lights|
199	409	2	21	target| targets|
200	1523	2	20	t test| fat test|
201	773	2	20	light| lightning|
202	1543	2	19	accord| accordance |
203	224	2	18	rate | rate climb|
204	1245	2	18	vibration| random vibration|
205	401	2	17	analysis | analysis plot|
206	1605	2	17	t trans|small unit transceiver|
207	859	2	16	ilsds| ilsds |
208	106	2	16	ilsds| ilsds |
209	108	2	16	ilsds| ilsds |
210	109	2	16	ilsds| ilsds |

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
211	1014	2	16	ilsds | ilsds
212	806	2	16	ilsds | ilsds
213	1342	2	15	solar | solar radiation
214	845	2	15	surface|metal
215	1384	2	15	solar | solar radiation
216	687	2	15	criteria| drop c
217	491	2	15	truck| trucks trailers
218	639	2	15	solar | solar radiation
219	177	2	15	truck| trucks trailers
220	1427	2	14	subscri| mtbf
221	1578	2	14	protection a| nfpa
222	779	2	13	finish| finish
223	1510	2	13	day night| n day
224	1580	2	13	organ| organic
225	1409	2	12	auto track|track
226	1175	2	12	ilsds |aging
227	183	2	12	d mobility| road mobility
228	986	2	12	circuit | circuit breakers
229	1579	2	12	navigation| navigation
230	987	2	12	circuit | circuit breakers
231	1415	2	11	auto |search
232	568	2	10	mtbmcf |f hour
233	818	2	9	desired| implementation
234	547	2	9	memory|implement
235	1624	2	9	aircraft| wing
236	471	2	7	gdt a| gdt attached gcs
237	1548	2	7	intelligence| intel
238	1549	2	7	input |input output
239	345	2	7	link | fade
240	1622	2	7	vision | revision m

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
361	686	3	21	rd te |radiation| radiation |
362	1367	3	18	vibration| vibration | vibration shock|
363	1092	3	18	safety |ty tests| safety tests|
364	170	3	18	comply | compl|f comp|
365	1085	3	18	qualifi| qualification | qualification tests|
366	1615	3	17	anned | unmanned | unmanned aerial |
367	1244	3	16	mobility | endurance| mobility e|
368	500	3	16	contained | self | self contained |
369	104	3	15	organization| concept| organization |
370	196	3	15	nuclear |biological chemical |nuclear biological chemical|
371	996	3	15	service| wiring | service|
372	1470	3	15	hours |t hours |attempt |
373	662	3	14	service| service | service s|
374	680	3	14	relative | humidity|humidity |
375	852	3	14	umentation| document| documentation|
376	1461	3	13	mtbf | mtbf m| mtbf mtbf |
377	1617	3	13	l trans|verse |universal transverse mercator|
378	872	3	12	e mmp | simulate | simulate m|
379	1144	3	11	rejection |retest| retest|
380	404	3	11	elements| plots|ts ele|
381	1480	3	10	istic |administrative logistic |t administrative |
382	1428	3	10	comput| compute| computed |
383	1412	3	10	sistance | jam resistance|resistance |
384	333	3	10	sistance|resistance| jam resistance|
385	1351	3	9	measurement|final c| measurements last hrs final cycle|
386	1032	3	9	inherent |ensure | lru inherent bit ensure bit met|
387	1025	3	9	inherent |ensure | lru inherent bit ensure bit met|
388	1581	3	9	radio|radio |net radio protoc|
389	193	3	9	survivability e|enhance| survivability enhancement|
390	1273	3	9	measurement|final c| measurements last hrs final cycle|

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
421	693	4	82	transpo| transpor| transport| transportability|
422	205	4	82	flight|program| preprogrammed | flight|
423	771	4	80	electro| electrostatic |discharge| electrostatic discharge esd |
424	114	4	80	performance | characteristics| performance character| performance
425	591	4	80	environment| environment| mode |environment |
426	236	4	77	flight| flight |mination | flight termination |
427	419	4	76	payload| payload | payload p|d plan|
428	1495	4	74	interface|sensor |or interface|sensor i|
429	528	4	72	processing |source|sources|resources|
430	1068	4	71	conditions|specti| condition| inspection con|
431	1213	4	71	recovery | recovery e|engine s| recovery en|
432	377	4	68	capability | comput| computers| computers c|
433	1552	4	68	support | support |integrated |integrated support plan|
434	1209	4	63	recovery| recovery| return | return recovery|
435	449	4	62	display |maintain | maintain | display |
436	617	4	61	flight| flight | icing| flight de|
437	1594	4	60	recovery |recovery ai| recovery |a reco|
438	179	4	54	power |cables| power c| power cables|
439	1490	4	54	console|operator co|operator con|operator console|
440	1488	4	53	display|location |location d| display|
441	857	4	51	training| personnel | personnel t| training|
442	735	4	51	design|k design| 2000 | soldering |
443	740	4	48	standard | standard p|ard part| standard parts|
444	1224	4	47	program| qualifi| qualification | qualification program|
445	34	4	47	power | power |ower ho|t power |
446	1477	4	46	airborne data terminal| adt airborne |dt airborne data | adt airbo
447	1112	4	44	function|functions|functions su| verification |
448	728	4	44	material|process|s process| materials processes|
449	1553	4	44	planning |information |strate|s joint s|
450	1047	4	42	respon|specti|inspection| inspection|

E.4 AbstFinder Results on the RFP Case Study: Last Iteration

```
        ***************************************************
        S U M M A R Y   OF      A B S T F I N D
        ***************************************************

-------------------------------------------------------
# of lines read from input file is 1912
# of abstractions found is 251
-------------------------------------------------------

------  |------ |------ |------ |--------------------
 #)     | abst# | corr_ | corr_ | correlateted-phrases
        |       |phras# |lines# |
------  |------ |------ |------ |--------------------
 1      | 10    | 1     | 6     | program|
 2      | 53    | 1     | 5     | amming|
 3      | 147   | 1     | 5     |  particular |
 4      | 148   | 1     | 5     |  partic|
 5      | 171   | 1     | 5     | mission |
 6      | 36    | 1     | 4     |  equal |
 7      | 39    | 1     | 4     | tability|
 8      | 112   | 1     | 4     |  manufacture|
 9      | 8     | 1     | 4     |  conce|
 10     | 226   | 1     | 4     |  equal |
 11     | 35    | 1     | 3     | board |
 12     | 1     | 1     | 3     |  compr|
 13     | 42    | 1     | 3     |  accom|
 14     | 47    | 1     | 3     |  assum|
 15     | 61    | 1     | 3     | board |
 16     | 69    | 1     | 3     | tinent|
 17     | 99    | 1     | 3     | rable |
 18     | 118   | 1     | 3     |  nonop|
 19     | 126   | 1     | 3     |  energ|
 20     | 133   | 1     | 3     |  compl|
 21     | 138   | 1     | 3     |  graphi|
 22     | 170   | 1     | 3     | function|
 23     | 205   | 1     | 3     |  appro|
 24     | 239   | 1     | 3     |  assum|
 25     | 246   | 1     | 3     |  position|
 26     | 250   | 1     | 3     |  recon|
 27     | 26    | 1     | 2     |  exhib|
 28     | 48    | 1     | 2     | planne|
 29     | 49    | 1     | 2     | minutes |
 30     | 52    | 1     | 2     | accept|
```

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
31	2	1	2	records
32	54	1	2	integ
33	55	1	2	succe
34	58	1	2	access
35	27	1	2	ensity
36	63	1	2	defin
37	64	1	2	defin
38	66	1	2	scan
39	67	1	2	scan
40	68	1	2	sequent
41	30	1	2	enhance
42	70	1	2	recogni
43	72	1	2	ident
44	77	1	2	ability
45	80	1	2	rection
46	82	1	2	mulate
47	85	1	2	derati
48	90	1	2	package
49	93	1	2	stand
50	95	1	2	ventilati
51	96	1	2	permi
52	31	1	2	servic
53	100	1	2	types
54	101	1	2	types
55	102	1	2	exhib
56	103	1	2	guide
57	105	1	2	appro
58	106	1	2	mounted
59	108	1	2	config
60	111	1	2	aded p

E.5 ASSR - Allocated System Software Requirements for the RFP

The following pages contain part of the Allocated System Software Requirements for the RFP generated by the human experts.

19.3. Software shall provide the capability to playback payload and AV data during autonomous operations or upon operator command.

3.7.1.2.6.f.

$ 3 MSSN _ COMM _ PYLD _ AP _ MCU _ GDT _ RVT _ SPRT

19.4. Software shall provide the capability to record payload data in case of loss of data link.

3.7.1.2.6.f.

$ 3 MSSN _ COMM _ PYLD _ AP _ MCU _ GDT _ RVT _ SPRT

19.5. Software shall provide the capability to record AV data in case of loss of data link.

20. 3.1.5 INTERFACE DEFINITION.

_ _ _ _

$ _ MSSN _ COMM _ PYLD X AP _ MCU _ GDT _ RVT _ SPRT

20.1. Software shall provide the capability to transmit data between GCS and TACFIRE (Removed by the Government as a requirement for the Phase I Contract).

_ _ _ _

$ _ MSSN _ COMM _ PYLD X AP _ MCU _ GDT _ RVT _ SPRT

20.2. Software shall provide the capability to receive data between GCS and TACFIRE (Removed by the Government as a requirement for the Phase I Contract).

_ _ _ _

$ _ MSSN _ COMM _ PYLD X AP _ MCU _ GDT _ RVT _ SPRT

20.3. Software shall provide the capability to transmit data between MPS and TACFIRE (Removed by the Government as a requirement for the Phase I Contract).

_ _ _ _ _

$ _ MSSN _ COMM _ PYLD X AP _ MCU _ GDT _ RVT _ SPRT

20.4. Software shall provide the capability to receive data between MPS and TACFIRE (Removed by the Government as a requirement for the Phase I Contract).

3.1.5.1.

$ _ MSSN _ COMM _ PYLD 7 AP _ MCU _ GDT _ RVT _ SPRT

20.5. Software shall provide the capability to transmit data between GCS and AFATDS, JSTARS GSM and GCRS/GPF.

3.1.5.1.

$ _ MSSN _ COMM _ PYLD 7 AP _ MCU _ GDT _ RVT _ SPRT

20.6. Software shall provide the capability to receive data between GCS and AFATDS, JSTARS GSM and GCRS/GPF.

3.1.5.1.

$ _ MSSN _ COMM _ PYLD 7 AP _ MCU _ GDT _ RVT _ SPRT

20.7. Software shall provide the capability to transmit data between MPS and AFATDS, JSTARS GSM and GCRS/GPF.

3.1.5.1.

$ _ MSSN _ COMM _ PYLD 7 AP _ MCU _ GDT _ RVT _ SPRT

20.8. Software shall provide the capability to receive data between MPS and AFATDS.

3.1.5.1.

$ _ MSSN _ COMM _ PYLD 7 AP _ MCU _ GDT _ RVT _ SPRT

20.9. Software shall provide the capability to transmit data between GCS and ASAS.

E.6 Sub-abstractions of Testing Abstraction (Summary)

```
        ***************************************************
        S U M M A R Y   OF      A B S T F I N D
        ***************************************************

-------------------------------------------------------
# of lines read from input file is 163
# of abstractions found is 160
-------------------------------------------------------

------  |------ |------ |------ |---------------------
 #)     | abst# | corr_ | corr_ | correlateted-phrases
        |       |phras# |lines# |
------  |------ |------ |------ |---------------------
 1      | 155   | 1     | 53    | test equipment|
 2      | 159   | 1     | 18    | t test|
 3      | 56    | 1     | 13    | acceptance tests|
 4      | 73    | 1     | 6     | retest|
 5      | 79    | 1     | 4     | test re|
 6      | 76    | 1     | 4     | 1 test|
 7      | 5     | 1     | 2     | minimum f|
 8      | 45    | 2     | 36    | tests |detailed test|
 9      | 157   | 2     | 18    | t test|built test|
 10     | 71    | 2     | 9     | corrective action |corrective action retest|
 11     | 42    | 2     | 9     | qualification |qualification tests|
 12     | 69    | 2     | 8     | rejection |retest|
 13     | 48    | 2     | 7     | ty tests|safety tests|
 14     | 140   | 2     | 6     | accordance |test levels |
 15     | 10    | 2     | 5     | test fa|test fat |
 16     | 63    | 2     | 5     | n test|burn test|
 17     | 158   | 3     | 67    | test equipment|t test |built test |
 18     | 160   | 3     | 59    | equipment|test measurement diagnostic equipment|e test |
 19     | 28    | 3     | 56    | test equipment |facilities|equipment f|
 20     | 150   | 3     | 41    | performing |tests |safety |
 21     | 7     | 3     | 27    | t test|built test|performing |
 22     | 122   | 3     | 22    | ct test |complete|impact test |
 23     | 39    | 3     | 22    | inspection|inspections tests|special inspections tests|
 24     | 149   | 3     | 15    | consist|environment|test levels |
 25     | 46    | 3     | 10    | environmental|environmental |environmental tests|
 26     | 78    | 3     | 5     | report|test re|test report|
 27     | 30    | 4     | 65    | test equipment |inspection |accordance |equipment acc|
 28     | 50    | 4     | 22    | t test |built test |demonstration|test d|
 29     | 145   | 4     | 19    | test s|flight |ain test |phases|
```

```
30     | 152   | 4     | 14    | operation|operational |operational te|operational test|
```

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
31	109	5	60	ct test |procedure|test p|test procedure|impact test |
32	2	5	57	equipment |equipment t|test measur|urement |test measurement diagno
33	27	5	49	tests |inspection|inspections tests |special inspections tests|spec
34	49	5	40	tests |design |tests demonstrate |safety |demonstrate s|
35	26	5	29	facilities|test fa|government |verification |government co|
36	115	5	26	tance |test c|test car|distance |e test car|
37	118	5	18	test c|test car|test car r|buffer cars|e test car|
38	113	5	17	test c|test car|test car |buffer cars |locomotive|
39	90	6	63	operati|equipment |storage |temperature |operating test temperature
40	153	6	40	tests |conducted |s conducted |tests con|tests conducted |cted f|
41	156	6	36	program|test p|acceptance test|test pro|test program|acceptance tes
42	24	6	34	test p|provisions |test plan|verification pro|s ident|compliance |
43	67	6	25	test a|approv|acceptance test|acceptance test |acceptance test appr
44	20	6	24	test b|t test |built test |mission |built test bit |built test bit
45	89	6	17	operati|training|temperature |t operati|operating test temperature
46	1	7	59	equipment |test equipment |equipment ma|d test |standard |rd test|r
47	32	7	51	test me|method|s procedure|methods |s procedures|ds pro|test method
48	123	7	51	rd test|tests |tests i|maintain |d tests|impact|step m|
49	60	7	43	performance|tests |conducted |design |s conducted |tests con|tests
50	16	7	40	tests |design|y tests |ty tests |humidity |tested |dity test|
51	61	7	30	program|test p|maintain|test pro|test program|support|maintainabili
52	138	7	28	exposure |s test|posed |examination|ts test|examination con|examina
53	116	7	25	tance |test c|commodity |test car|test car |buffer cars|distance |
54	14	7	25	cluding |test b|t test |built test |quantit|comply |built test bit
55	18	7	24	test b|t test |built test |mission |built test bit |mission critica
56	41	7	24	st insp|test i|inspection |t inspection|compliance |mainta|tained |
57	31	8	64	test equipment |1 test|inspection |equipment qu|accuracy |quality |
58	4	8	64	test equipment |equipment t|control|environment|facilities|environm
59	97	8	60	equipment |test s|withstand |equipment sub|subjected |herein|equipm
60	17	8	59	equipment |equipment t|s equipment |ground |provisions |ds pro|equi

E.7 Sub-abstractions of Safety and Fail Abstractions

```
        ***************************************************
        S U M M A R Y   OF     A B S T F I N D
        ***************************************************

-----------------------------------------------------
# of lines read from input file is 15
# of abstractions found is 14
-----------------------------------------------------

------  |------ |------ |------ |---------------------
 #)     | abst# | corr_ | corr_ | correlateted-phrases
        |       |phras# |lines# |
------  |------ |------ |------ |---------------------
 1      | 3     | 1     | 15    | safety|
 2      | 5     | 1     | 15    | safety |
 3      | 6     | 1     | 15    | safety |
 4      | 12    | 1     | 15    | safety |
 5      | 2     | 2     | 15    | safety |safety f|
 6      | 10    | 2     | 15    | safety |safety tests|
 7      | 14    | 2     | 15    | safety |safety h|
 8      | 11    | 3     | 15    | safety |tests |demonstrate|
 9      | 13    | 3     | 15    | safety |tests |perform|
 10     | 4     | 3     | 15    | safety |safety p|safety f|
 11     | 1     | 4     | 15    | flight |safety |flight termination |safety h|
 12     | 7     | 4     | 15    | safety |qualification environmental safety te|safety tests|tests |
 13     | 0     | 4     | 15    | flight |safety |flight termination |safety p|
 14     | 8     | 5     | 15    | safety |qualification environmental safety te|qualification enviror
```

```
        ***************************************************
        S U M M A R Y   OF     A B S T F I N D
        ***************************************************

-------------------------------------------------------
# of lines read from input file is 46
# of abstractions found is 45
-------------------------------------------------------

------  |------ |------ |------ |---------------------
 #)     | abst# | corr_ | corr_ | correlateted-phrases
        |       |phras# |lines# |
------  |------ |------ |------ |---------------------
 1      | 36    | 2     | 44    | failure |mean time mission critical failure mtbmcf |
 2      | 19    | 2     | 43    | failure |failure reporting|
 3      | 5     | 2     | 43    | failure|personnel |
 4      | 37    | 3     | 44    | failure|mission |mission failure|
 5      | 45    | 3     | 44    | failure|mtbmcf |mean time mission critical failure|
 6      | 14    | 4     | 44    | failure|mission critical failure|mission critical failures|built te
 7      | 9     | 5     | 44    | failure|mission critical failure|mission critical failures|built te
 8      | 42    | 5     | 44    | failure |failure r|system|subsystem|subsystem |
 9      | 20    | 5     | 43    | failure |failures|failures |testing |s occur|
 10     | 4     | 6     | 45    | failure|function |s fail|s failure|function result|operate |
 11     | 2     | 6     | 44    | failure |mission |mission critical failure|mean time mission critic
 12     | 16    | 6     | 43    | failure|failures|failures |t isolate |bit isolate |failures single
 13     | 31    | 7     | 45    | failure |mission critical failure |s mission critical failure|syste
 14     | 1     | 7     | 44    | recovery |recovery d|failure|event |manual |ency re|catastrophic |
 15     | 6     | 7     | 44    | damage |failure |protection d|power |equipment |power failure|failu
 16     | 24    | 8     | 45    | failure |s failure|failure da|testing |corrective action |ss fail|a
 17     | 34    | 8     | 45    | failure|mission |failures|failures |t mission |mission failures |s
 18     | 25    | 8     | 44    | failure pr|deficiencies |deficiencies d|corrective action |criteria
 19     | 32    | 8     | 44    | failure |failure mtb|mean time |mission |e failure|reliability |mea
 20     | 7     | 8     | 44    | damage |failure|function |protection d|power |equipment |power fail
 21     | 3     | 9     | 45    | failure|mission |mission critical failure|failures|mission critical
 22     | 12    | 9     | 44    | failure|mission critical failure|mission critical failures|failures
 23     | 33    | 9     | 44    | failure|manual|mission |failures|failures |technical |mission failu
 24     | 11    | 9     | 44    | failure|mission critical failure|mission critical failures|failures
 25     | 23    | 9     | 15    | establ|s fail|ss fail|criteria |systems|control |components |s comp
 26     | 17    | 10    | 44    | failure|mission |failures|failures |isolat|percent mission |additio
 27     | 0     | 10    | 44    | recovery |recovery d|failure|event |failure |damage |maintenance |c
 28     | 39    | 10    | 44    | failure |mission |s mission |function|function result|malfunction |
 29     | 30    | 11    | 45    | failure |mission |s mission |s failure|components |ss fail|ess fail
 30     | 22    | 11    | 44    | failure |ency re|addition|failure reporting|deficienc|timely |failu
```

#)	abst#	corr_phras#	corr_lines#	correlateted-phrases
31	15	11	44	failure|mission critical failure|mission critical failures|equipmer
32	28	11	7	t fail|uring |s proc|activity |activity de|procuring activity |rete
33	10	12	44	failure|mission critical failure|mission critical failures|equipmer
34	26	12	44	failure pr|s appro|uring |activity |corrective action |failure pro
35	18	12	43	failure|manual |failures|failures |isolat|t isolat|isolation |bit i
36	35	13	44	maintenance |failure|mission |failures|l mission |personnel |equipr
37	13	13	44	failure|manual |mission critical failure|mission critical failures
38	40	14	45	failure |failure mtb|mean time |failures|e failure|failures |e fail
39	41	14	45	failure|mission |failures|operate |failures |t mission |mission fai
40	27	15	46	failure |t failure|uring |defects |activity de|respons|correcti|inv
41	43	15	45	failure|mission |failures|operate |failures |t mission |failures i|
42	21	15	44	design|failure |detect |proced|failure reporting|deficienc|timely |
43	8	16	45	maintenance |failure|mission critical |failures|equipment |failures
44	38	16	44	failure |catastrophic |mission |critical |function |personnel |equi
45	44	19	36	recovery |mission |s fail|equipment |launch |e fail|t fail|t missio

E.8 Results of findphrases on the RFP Case Study

```
**********   268    equipment
*********    252    /
*******      195    data
******       155    mission
*****        143    mil -
*****        138    test
*****        130    - sr
*****        130    uav -
*****        130    uav - sr
****         112    - std
****         110    - std -
****         110    std -
****         107    |
****         100    ground
****          98    control
****          97    mil - std
****          97    the uav
****          97    the uav -
****          97    the uav - sr
****          95    mil - std -
***           90    - sr system
***           90    sr system
***           90    uav - sr system
***           89    the system
***           70    mpcs
***           69    performance
***           67    the av
**            66    payload
**            66    time
**            63    power
**            62    recovery
**            59    mmp
**            54    flight
**            53    government
**            52    launch
**            51    equipment shall
**            50    maintenance
**            50    support
**            49    operator
**            48    tests
**            45    shall be capable of
**            44    storage
**            43    electrical
```

```
**          43    standard
**          43    the equipment
**          41    components
**          41    gcs
*           40    gdt
*           39    ) ,
*           39    , the
*           39    - 810d
*           39    - std - 810d
*           39    link
*           39    std - 810d
*           39    the mpcs
*           38    be provided
*           38    mps
*           38    of mil
*           38    of mil -
*           37    environmental
*           37    method
*           36    contractor
*           36    l
*           35    imint
*           35    relay
*           35    subsystem
*           34    data link
*           34    electromagnetic
*           34    of mil - std
*           34    station
*           32    air vehicle
*           32    day
*           32    design
*           32    planning
*           32    see appendix
*           32    temperature
*           31    reliability
*           31    shelters
*           31    video
*           30    command
*           30    the contractor
*           30    training
*           29    conditions
*           29    r
*           29    sensor
*           29    terminal
*           28    - sr system shall
*           28    / r
```

```
*            28    be subjected
*            28    interface
*            28    l /
*            28    l / r
*            28    non -
*            28    procedure
*            28    testing
*            27    airborne
*            27    appendix 100
*            27    display
*            27    failure
*            27    may be
*            27    total
*            26    ) 3
*            26    environments
*            26    mission planning
*            26    on the
*            26    operate
*            26    personnel
*            26    shall be subjected to
*            26    the government
*            25    , shall
*            25    100 )
*            25    appendix 100 )
*            25    phase
*            25    speed
*            25    status
*            25    the system shall
*            24    adr
*            24    digital
*            24    electronic
*            24    growth
*            24    parts
*            24    requirements of
*            24    see appendix 100
*            24    subsystems
*            24    transport
*            23    bit
*            23    degradation
*            23    modular
*            23    to the test
*            22    ( confidential
*            22    , see
*            22    , see appendix
*            22    be subjected to the
```

```
*            22    characteristics
*            22    confidential
*            22    equipment ,
*            22    military
*            22    see appendix 100 )
*            22    support equipment
*            22    test of
*            22    the capability
*            22    the equipment shall
*            22    unit
*            22    x
*            22    | |
*            21    %
*            21    ( confidential ,
*            21    ( confidential , see
*            21    ) (
*            21    +
*            21    , see appendix 100
*            21    a minimum
*            21    acceptance
*            21    airborne data
*            21    as a
*            21    confidential ,
*            21    confidential , see
*            21    confidential , see appendix
*            21    failures
*            21    hours
*            21    imagery
*            21    installed
*            21    meet the
*            21    of the system
*            21    program
*            21    system shall be
*            21    the test of
*            21    to the test of
*            21    x |
*            21    | x
*            21    | x |
*            20    ( s
*            20    ( s )
*            20    - 1
*            20    car
*            20    damage
*            20    facilities
*            20    imint mmp
```

```
*            20    procedures
*            20    quality
*            20    s )
*            20    specified in
*            20    subjected to the test
*            20    the equipment shall be
*            20    the requirements
*            20    with a
*            19    1 )
*            19    and recovery
*            19    at least
*            19    av and
*            19    capability to
*            19    data terminal
*            19    equipment shall be subjected
*            19    location
*            19    modular mission
*            19    of method
*            19    para 3
*            19    part
*            19    rate
*            19    shelter
*            19    the data
*            18    , shall be
*            18    / hr
*            18    2 )
*            18    accordance
*            18    accordance with
*            18    at the
*            18    board
*            18    c -
*            18    conducted
*            18    fault
*            18    in accordance
*            18    in accordance with
*            18    inspection
*            18    inspections
*            18    mtbf
*            18    on -
*            18    processing
*            18    qualification
*            18    rvt
*            17    "
*            17    ( 1
*            17    ( 2
```

*	17	2 ,
*	17	3)
*	17	area
*	17	be conducted
*	17	control station
*	17	criteria
*	17	critical
*	17	data relay
*	17	fungus
*	17	is required to
*	17	level
*	17	rain
*	17	selectable
*	17	test of method
*	17	the test of method
*	17	this specification
*	17	transit
*	17	used to
*	16	’
*	16	(1)
*	16	) shall
*	16	- c
*	16	- |
*	16	4)
*	16	70 -
*	16	=
*	16	communications
*	16	downlink
*	16	environment
*	16	ilsds
*	16	interfaces
*	16	mission payload
*	16	payloads
*	16	physical
*	16	production
*	16	remote
*	16	safety
*	16	shall meet
*	16	shall operate
*	16	station (
*	16	the capability to
*	16	the mission
*	16	the mps
*	16	use of
*	16	vibration

```
*            16    weight
*            16    wind
*            16    within the
*            16    withstand
*            16    | -
*            15    ( 2 )
*            15    - board
*            15    - in
*            15    altitude
*            15    and control
*            15    and shall be
*            15    approved
*            15    bus
*            15    communication
*            15    consist of
*            15    corrective
*            15    ft
*            15    induced
*            15    launch and
*            15    of the uav
*            15    of the uav -
*            15    on - board
*            15    on a
*            15    operating ,
*            15    part of
*            15    primary
*            15    real
*            15    recorder
*            15    shall include
*            15    shall meet the
*            15    shock
*            15    such as
*            15    tactical
*            15    target
*            15    terminal (
*            15    the air
*            15    the operator
*            15    the performance
*            15    work
*            14    ' s
*            14    - c -
*            14    21
*            14    activity
*            14    adt
*            14    and other
```

```
*             14    coating
*             14    common
*             14    compliance
*             14    contract
*             14    contractor shall
*             14    devices
*             14    except
*             14    external
*             14    fat
*             14    ground data
*             14    is the
*             14    materials
*             14    mil - c
*             14    mil - c -
*             14    mobile
*             14    must
*             14    of this
*             14    performed
*             14    prior to
*             14    radiation
*             14    shall consist
*             14    shall consist of
*             14    specified performance
*             14    the requirements of
*             14    with mil
*             14    with mil -
```

Appendix F Using Abstraction Network of Findphrases Case Study

Figure 15 contains snapshots of using the abstraction network of the findphrases case study. Figure 15-a is the COVER of the database seen when entering the browser of Hyperties. On the top of the screen appears the name of the database, *FINDPHRASES ABSTRACTION NETWORK*. The user can *Enter Knowledge Base*, or get information *About This Knowledge Base*, or call for *Help*. Figure 15-b is the INTRODUCTION screen after entering the database from the COVER. This screen was created by the author, and is a list of all the abstractions of findphrases. The INDEX window in Figure 15-e is generated automatically by Hyperties and contains also the list of findphrases abstractions (titles), plus all the technical articles connected to the visual display (Hyperties defaults). The user, looking at the INTRODUCTION screen, chooses one of the abstractions, for example text_file, and gets a description screen about that abstraction article (See Figure 15-c).

Then, he or she can *remove* it if it is not interesting, or choose to *Read This Article*. Figure 15-d shows the content of the *TextFile* abstraction article. The keywords identifying the other abstractions, e.g., "phrases", "symbolcharacters", "output", "punctuation", etc., are the automatically generated links to other abstractions and are in a color different from than of the text. So, the reader may choose to read the text sequentially by using *Next pg*, or to follow the links associatively. On the bottom of the screen, there are user buttons: *Backup* for backtracking according to the HISTORY Screen (See Figure 15-f), *Search* for any string, *Help*, *Index*, and *Exit*. Upon pushing *Index*, the user gets the INDEX screen (See Figure 15-e). On the INDEX screen, there is the *History* button used for getting the HISTORY screen (See Figure 15-f) in order to see the reader's path in case he or she gets lost. Hyperties always keeps the history of the navigation path and enables the user to backtrack in this path (See Appendix F.2).

Table 4 shows the links in the findphrases abstraction network, which were automatically generated according to the actions described previously of the second stage of REGAE. Those automatically generated links are actually cross-reference-type links, as suggested by the REGAE flight-passenger example in Section 1.5. Almost every abstraction is connected to almost every other. Most of the connections are bi-directional. Thus, the graphic description of this hypertext is really a web and will not help solve the orientation problem. Since the version of Hyperties used does not permit adding new designs, a graphic decomposition of the network cannot be generated. From the use of this example, it is clear that a graphic description such as shown in Figure 14 in Appendix C.4 is desirable.

It is the belief of this author that the nature of the diagram in a requirements environment be completely unspecified. The elicitor should be free to draw whatever diagram he or she believes will be meaningful to the client.

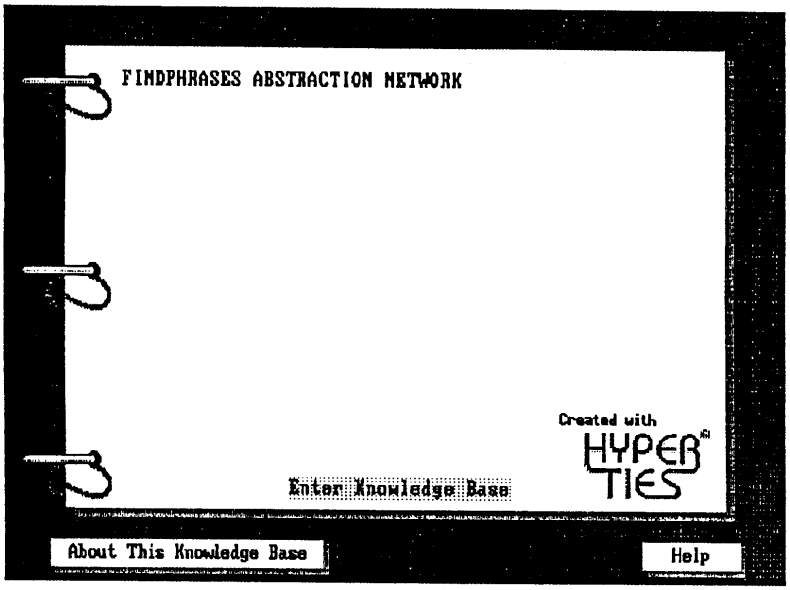
After constructing the abstraction network, five sentences were found that are not contained in any abstraction. Most of them were not important, and did not give new information. Only two of them were connected to the text by context, such as "Consequently, ...", and they did not give any new information too. Just to be on the safe side, these sentences were collected to a garbage collector abstraction. Moreover, the original transcript was kept in the text nodes,

with pointers to and from the sentences of the abstraction nodes, in order to be able to examine sentences of abstractions in their original context.

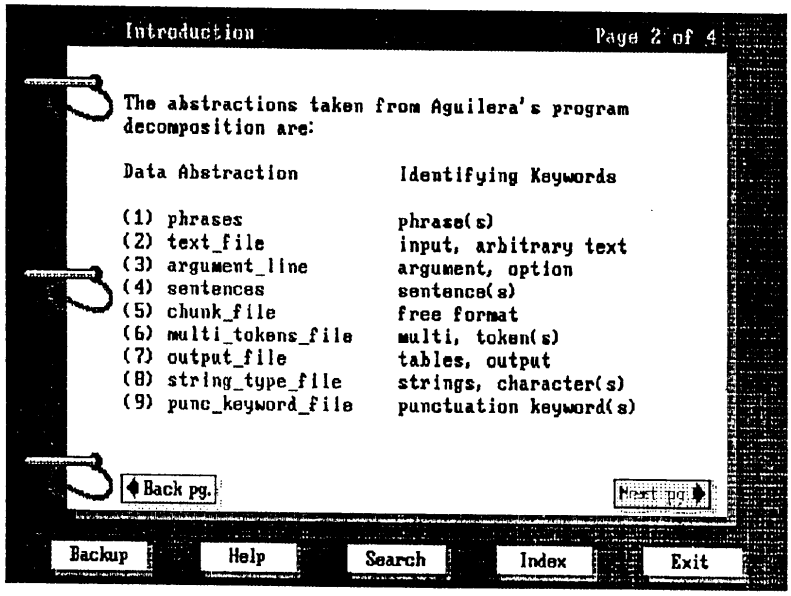
The abstraction is linked to:	(1)	(2)	(3)	(4)	(5)	(6)	(7)	(8)	(9)
(1) phrases		x	x	x		x	x		x
(2) text_file	x						x	x	x
(3) argument_line	x	x		x	x	x	x	x	x
(4) sentences	x					x	x	x	x
(5) chunk_file						x		x	x
(6) multi_token	x		x	x	x		x	x	x
(7) output_file	x	x	x	x		x		x	x
(8) string_file		x		x	x	x			x
(9) punctuation	x	x		x	x	x	x	x	

Table 4: The links generated automatically by AbstFinder.
טבלה 4: הקשרים שנוצרו אוטומטית על ידי AbstFinder

Figure 15: Hyperties with findphrases
findphrases עם Hyperties :15 ציור

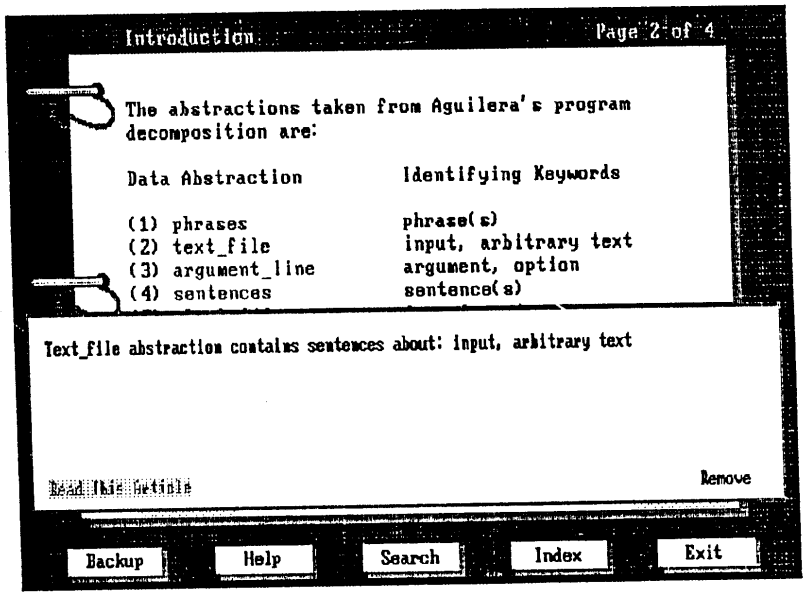


a

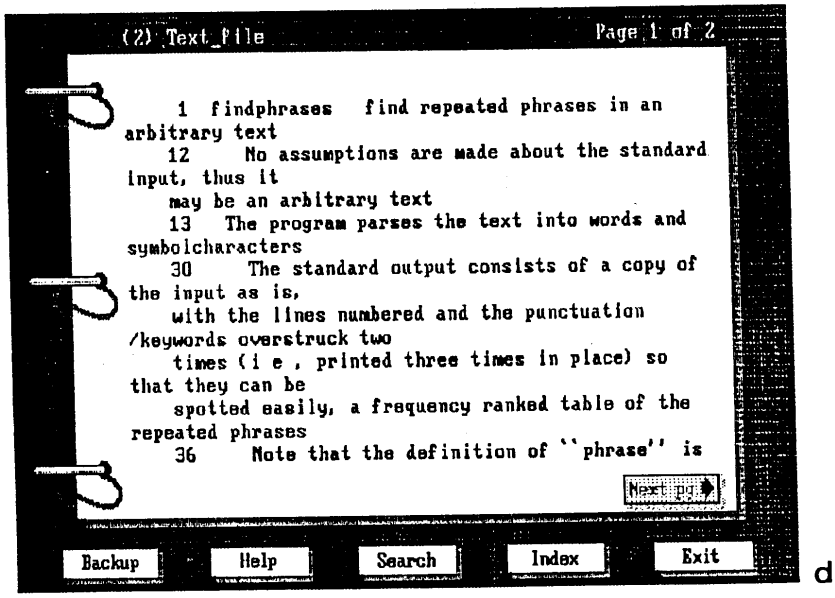


b

Figure 15: Hyperties with findphrases (Cont.)
findphrases עם Hyperties :15 ציור

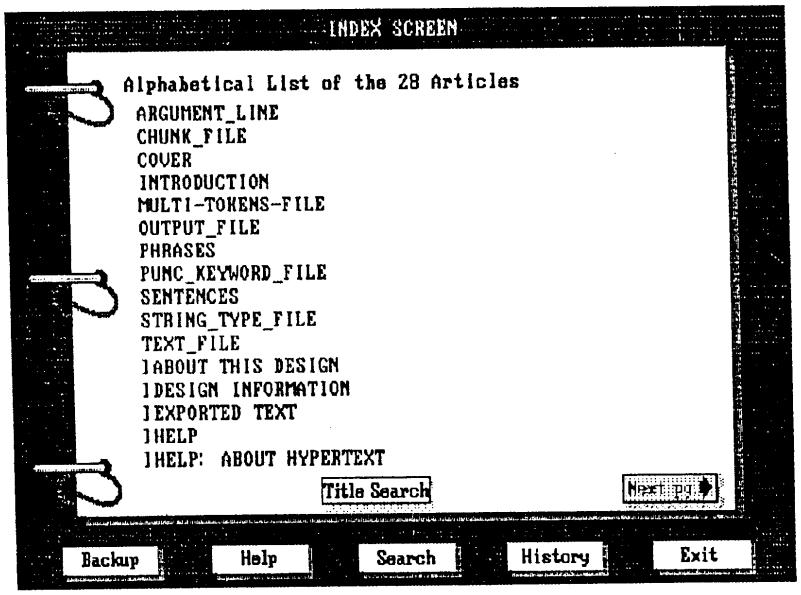


c

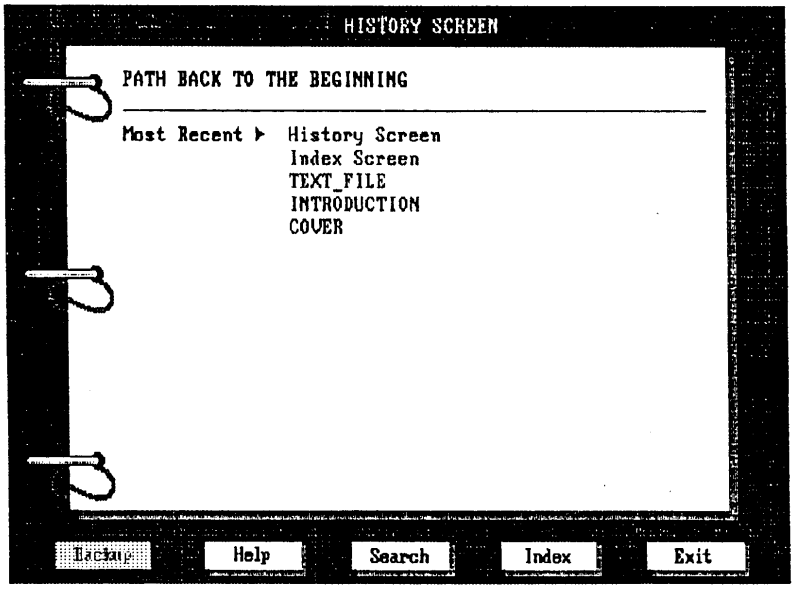


d

Figure 15: Hyperties with findphrases (Cont.)
findphrases עם Hyperties :15 ציור



e



f

F.1 Makefile for generating links automatically

```
#-------------------------------------------------------------------
# makefile: for generate automatically links identified abstractions
#           for Hyperties (according to Hyperties mark-up language).
#-------------------------------------------------------------------

#      macro definitions

SOURCES    =

#      target definitions

a1.phrase: abst.1.phrase abst.Ph.1
# remove TABs
       fmt -s abst.1.phrase > a1
# preapare links of <L> phrase <
       sed -f abst.Ph.1 a1 > a1.phrase
       rm a1

a2.text: abst.2.text abst.Ph.2
# remove TABs
       fmt -s abst.2.text > a2
# preapare links of <L> phrase <
       sed -f abst.Ph.2 a2 > a2.text
       rm a2

a3.argu: abst.3.argu abst.Ph.3
# remove TABs
       fmt -s abst.3.argu > a3
# preapare links of <L> phrase <
       sed -f abst.Ph.3 a3 > a3.argu
       rm a3

a4.sentenc: abst.4.sentenc abst.Ph.4
# remove TABs
       fmt -s abst.4.sentenc > a4
# preapare links of <L> phrase <
       sed -f abst.Ph.4 a4 > a4.sentenc
       rm a4

a5.chunk: abst.5.chunk abst.Ph.5
# remove TABs
       fmt -s abst.5.chunk > a5
# preapare links of <L> phrase <
```

```
	sed -f abst.Ph.5 a5 > a5.chunk
	rm a5

a6.token: abst.6.token abst.Ph.6
# remove TABs
	fmt -s abst.6.token > a6
# preapare links of <L> phrase <
	sed -f abst.Ph.6 a6 > a6.token
	rm a6

a7.output: abst.7.output abst.Ph.7
# remove TABs
	fmt -s abst.7.output > a7
# preapare links of <L> phrase <
	sed -f abst.Ph.7 a7 > a7.output
	rm a7

a8.string: abst.8.string abst.Ph.8
# remove TABs
	fmt -s abst.8.string > a8
# preapare links of <L> phrase <
	sed -f abst.Ph.8 a8 > a8.string
	rm a8

a9.punc: abst.9.punc abst.Ph.9
# remove TABs
	fmt -s abst.9.punc > a9
# preapare links of <L> phrase <
	sed -f abst.Ph.9 a9 > a9.punc
	rm a9

#-----------------------------------------------------------------------
# CorrPhrases.abst: contains the phrases that identify the abstractions
#-----------------------------------------------------------------------
abst.puncuation:  punctuation keyword file|
abst.phrases:     phrase | phrases
abst.argument:    argument | optional |
abst.string:      character|symbolcharacter|string
abst.chunk:       free format |
abst.text:        arbitrary text|input |
abst.muli-token   multi tokens file|
abst.output:      output| tables |tables output|
abst.sentence:    sentence |

#-----------------------------------------------------------------------
```

```
# Abst.Ph.2: contains the script for generating links in abst.text (2)
#-------------------------------------------------------------------
s/ punctuation/ <L>punctuation<\L>/g
s/ Punctuation/ <L>Punctuation<\L>/g
s/ phrases/ <L>phrases<\L>/g
s/ Phrases/ <L>Phrases<\L>/g
s/ phrase/ <L>phrase<\L>/g
s/ Phrase/ <L>Phrase<\L>/g
s/ argument/ <L>argument<\L>/g
s/ Argument/ <L>Argument<\L>/g
s/ arguments/ <L>arguments<\L>/g
s/ Arguments/ <L>Arguments<\L>/g
s/ symbolcharacters/ <L>symbolcharacters<\L>/g
s/ Symbolcharacters/ <L>Symbolcharacters<\L>/g
s/ symbolcharacter/ <L>symbolcharacter<\L>/g
s/ Symbolcharacter/ <L>Symbolcharacter<\L>/g
s/ characters/ <L>characters<\L>/g
s/ Characters/ <L>Characters<\L>/g
s/ character/ <L>character<\L>/g
s/ strings/ <L>strings<\L>/g
s/ Strings/ <L>Strings<\L>/g
s/ string/ <L>string<\L>/g
s/ String/ <L>String<\L>/g
s/ free format/ <L>free format<\L>/g
s/ Free format/ <L>Free format<\L>/g
s/ multi-tokens-file/ <L>multi-tokens-file<\L>/g
s/ Multi-tokens-file/ <L>Multi-tokens-file<\L>/g
s/ multi-tokens/ <L>multi-tokens<\L>/g
s/ Multi-tokens/ <L>Multi-tokens<\L>/g
s/ multi-token/ <L>multi-token<\L>/g
s/ Multi-token/ <L>Multi-token<\L>/g
s/ tokens/ <L>tokens<\L>/g
s/ Tokens/ <L>Tokens<\L>/g
s/ token/ <L>token<\L>/g
s/ Token/ <L>Token<\L>/g
s/ tables output/ <L>tables output<\L>/g
s/ Tables output/ <L>Tables output<\L>/g
s/ output/ <L>output<\L>/g
s/ Output/ <L>Output<\L>/g
s/ tables/ <L>tables<\L>/g
s/ Tables/ <L>Tables<\L>/g
s/ table/ <L>table<\L>/g
s/ Table/ <L>Table<\L>/g

#-------------------------------------------------------------------
```

```
# a2.text: is the abstraction network node (article) for abst.text
#          after activating makefile.link
#-----------------------------------------------------------------
     1  findphrases   find repeated <L>phrases<
    12     No assumptions are made about the standard input, thus it
    may be an arbitrary text
    13   The program parses the text into words and <L>symbolcharacters<
    30     The standard <L>output<
    with the lines numbered and the <L>punctuation<
    times (i e , printed three times in place) so that they can be
    spotted easily, a frequency ranked <L>table<phrases<
    36     Note that the definition of ‘‘phrase’’ is independent of the
    number of times it occurs in the sentences
```

F.2 Example of using Hyperties with logging (findphrases)

```
Wed Sep 9, 1992 16:02:57 BROWSER START => FPHRNET
Wed Sep 9, 1992 16:02:58 ARTICLE ENTRY => COVER
Wed Sep 9, 1992 16:03:03 ARTICLE ENTRY => INTRODUCTION
Wed Sep 9, 1992 16:03:28 ARTICLE ENTRY => CHUNK_FILE
Wed Sep 9, 1992 16:04:01 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:37:54 BROWSER START => FPHRNET
Sun Sep 13, 1992 18:37:54 ARTICLE ENTRY => COVER
Sun Sep 13, 1992 18:37:59 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:38:31 ARTICLE ENTRY => PHRASES
Sun Sep 13, 1992 18:41:08 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:41:29 ARTICLE ENTRY => TEXT_FILE
Sun Sep 13, 1992 18:43:37 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:43:51 ARTICLE ENTRY => ARGUMENT_LINE
Sun Sep 13, 1992 18:45:41 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:45:51 ARTICLE ENTRY => SENTENCES
Sun Sep 13, 1992 18:46:34 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:47:05 ARTICLE ENTRY => CHUNK_FILE
Sun Sep 13, 1992 18:47:42 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:47:51 ARTICLE ENTRY => MULTI-TOKENS-FILE
Sun Sep 13, 1992 18:49:18 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:49:30 ARTICLE ENTRY => OUTPUT_FILE
Sun Sep 13, 1992 18:50:48 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:50:59 ARTICLE ENTRY => STRING_TYPE_FILE
Sun Sep 13, 1992 18:52:07 ARTICLE ENTRY => INTRODUCTION
Sun Sep 13, 1992 18:52:15 ARTICLE ENTRY => PUNC_KEYWORD_FILE
Mon Sep 14, 1992 14:30:39 BROWSER START => FPHRNET
Mon Sep 14, 1992 14:30:39 ARTICLE ENTRY => COVER
Mon Sep 14, 1992 14:30:45 ARTICLE ENTRY => INTRODUCTION
Mon Sep 14, 1992 14:32:17 ARTICLE ENTRY => PHRASES
Mon Sep 14, 1992 14:32:29 ARTICLE ENTRY => INTRODUCTION
Mon Sep 14, 1992 14:32:38 ARTICLE ENTRY => PHRASES
Mon Sep 14, 1992 14:33:20 ARTICLE ENTRY => PUNC_KEYWORD_FILE
Mon Sep 14, 1992 14:33:40 ARTICLE ENTRY => PHRASES
Mon Sep 14, 1992 14:33:51 ARTICLE ENTRY => INTRODUCTION
Mon Sep 14, 1992 14:34:21 ARTICLE ENTRY => PHRASES
Mon Sep 14, 1992 14:34:47 ARTICLE ENTRY => INDEX
Mon Sep 14, 1992 14:38:15 ARTICLE ENTRY => OUTPUT_FILE
Mon Sep 14, 1992 14:47:17 ARTICLE ENTRY => TEXT_FILE
Mon Sep 14, 1992 14:47:33 ARTICLE ENTRY => OUTPUT_FILE
Mon Sep 14, 1992 14:47:48 ARTICLE ENTRY => INDEX
Mon Sep 14, 1992 14:48:28 ARTICLE ENTRY => ]HELP
Mon Sep 14, 1992 14:48:32 ARTICLE ENTRY => INDEX
Mon Sep 14, 1992 14:48:37 ARTICLE ENTRY => PHRASES
```

```
Mon Sep 14, 1992 14:48:44 ARTICLE ENTRY => INTRODUCTION
Mon Sep 14, 1992 14:48:59 ARTICLE ENTRY => COVER
Mon Sep 14, 1992 14:49:03 ARTICLE ENTRY => INTRODUCTION
Sun Sep 20, 1992 18:48:20 BROWSER START => FPHRNET
Sun Sep 20, 1992 18:48:20 ARTICLE ENTRY => COVER
Sun Sep 20, 1992 18:48:28 ARTICLE ENTRY => INTRODUCTION
Sun Sep 20, 1992 18:49:32 ARTICLE ENTRY => TEXT_FILE
Sun Sep 20, 1992 18:49:44 ARTICLE ENTRY => INDEX
Sun Sep 20, 1992 18:49:55 ARTICLE ENTRY => TEXT_FILE
Sun Sep 20, 1992 18:50:53 ARTICLE ENTRY => SEARCH
Sun Sep 20, 1992 18:51:03 ARTICLE ENTRY => TEXT_FILE
Sun Sep 20, 1992 18:51:10 ARTICLE ENTRY => INDEX
Wed Jan 6, 1993 14:24:40 BROWSER START => FPHRNET
Wed Jan 6, 1993 14:24:40 ARTICLE ENTRY => COVER
Wed Jan 6, 1993 14:26:48 ARTICLE ENTRY => INTRODUCTION
Sat Jan 16, 1993 00:30:30 BROWSER START => FPHRNET
Sat Jan 16, 1993 00:30:30 ARTICLE ENTRY => COVER
Sat Jan 16, 1993 00:30:34 ARTICLE ENTRY => INTRODUCTION
Sat Jan 16, 1993 00:54:29 BROWSER START => FPHRNET
Sat Jan 16, 1993 00:54:30 ARTICLE ENTRY => COVER
Sat Jan 16, 1993 00:54:35 ARTICLE ENTRY => INTRODUCTION
Sat Jan 16, 1993 00:59:30 BROWSER START => FPHRNET
Sat Jan 16, 1993 00:59:30 ARTICLE ENTRY => COVER
Sat Jan 16, 1993 00:59:35 ARTICLE ENTRY => INTRODUCTION
Sat Jan 16, 1993 01:05:54 ARTICLE ENTRY => TEXT_FILE
Sat Jan 16, 1993 01:06:17 ARTICLE ENTRY => INDEX
Sat Jan 16, 1993 01:06:34 ARTICLE ENTRY => HISTORY
Sat Jan 16, 1993 01:11:22 ARTICLE ENTRY => INDEX
Sat Jan 16, 1993 04:07:18 ARTICLE ENTRY => PUNC_KEYWORD_FILE
Sat Jan 16, 1993 04:08:24 ARTICLE ENTRY => INDEX
Sat Jan 16, 1993 04:08:39 ARTICLE ENTRY => TEXT_FILE
Sat Jan 16, 1993 04:08:45 ARTICLE ENTRY => INTRODUCTION
Sat Jan 16, 1993 04:08:50 ARTICLE ENTRY => TEXT_FILE
Sat Jan 16, 1993 04:14:37 BROWSER START => FPHRNET
Sat Jan 16, 1993 04:14:37 ARTICLE ENTRY => COVER
Sat Jan 16, 1993 04:20:45 ARTICLE ENTRY => INTRODUCTION
Sat Jan 16, 1993 04:20:59 ARTICLE ENTRY => TEXT_FILE
Tue Jan 19, 1993 02:12:21 BROWSER START => FPHRNET
Tue Jan 19, 1993 02:12:21 ARTICLE ENTRY => COVER
Tue Jan 19, 1993 02:12:28 ARTICLE ENTRY => INTRODUCTION
Tue Jan 19, 1993 02:30:51 BROWSER START => FPHRNET
Tue Jan 19, 1993 02:30:51 ARTICLE ENTRY => COVER
Tue Jan 19, 1993 02:30:56 ARTICLE ENTRY => INTRODUCTION
Sun Jan 31, 1993 01:00:01 BROWSER START => FPHRNET
Sun Jan 31, 1993 01:00:01 ARTICLE ENTRY => COVER
```

```
Sun Jan 31, 1993 01:00:29 ARTICLE ENTRY => INTRODUCTION
Sun Jan 31, 1993 01:40:56 ARTICLE ENTRY => TEXT_FILE
Sun Jan 31, 1993 01:53:59 ARTICLE ENTRY => INDEX
Sun Jan 31, 1993 02:08:07 ARTICLE ENTRY => HISTORY
Sun Jan 31, 1993 02:28:08 BROWSER START => FPHRNET
Sun Jan 31, 1993 02:28:08 ARTICLE ENTRY => COVER
Sun Jan 31, 1993 03:06:02 ARTICLE ENTRY => INTRODUCTION
```

F.3 Hyperties **input file with mark-up** (findphrases)

In this appendix, long lines are folded at a convenient word break. The break is denoted by a "\" at the end of all pieces of the line but the last and a space at the beginning of all pieces except the first.

```
<IE>ARGUMENT_LINE<\IE>
<H>(3) Argument_line<\H>
<D>Argument_line abstraction contains sentences about: argument, option<\D>
<C>
     6     The   n  argument is optional and if present provides a
     number  number  serving as the maximum length <L>phrase<\L=PHRASES> (to be
     described later) to be tallied
     7   If this argument is not present, if it does not supply a
     number, or if the supplied number is outside the reasonable range
     of greater than zero and less than or equal to 50, then  number
     is taken as 10
     9   The optional  ignored-phrases-file  contains one-per-line a
     list of <L>phrases<\L=PHRASES> to be ignored in the tallying (see below)
    11   The optional  <L>multi-tokens-file<\L=MULTI-TOKENS-FILE>  contains in <L>free format<\L=CHUNK_FILE> a
    list of those <L>character strings<\L=STRING_TYPE_FILE> consisting of more than one
    <L>symbolcharacter<\L=STRING_TYPE_FILE> (see below) which are to be taken as <L>multi-tokens<\L=MULTI-TOKENS-FILE>
    (see below)
    12     No assumptions are made about the standard <L>input<\L=TEXT_FILE>, thus it
    may be an arbitrary <L>text<\L=TEXT_FILE>
    14   These in turn are formed and classified into <L>tokens<\L=MULTI-TOKENS-FILE> and
    <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords based on the information provided by the
    <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  and, when the   m  option is present,
    the  <L>multi-tokens-file<\L=MULTI-TOKENS-FILE>
    29   If the   b  option is used along with the
    ignored-phrases-file, then <L>phrases<\L=PHRASES> which begin with an ignored
    <L>phrase<\L=PHRASES> are also ignored in the tallying
```

```
    39   When the   b  option is present, a <L>phrase<\L=PHRASES> which begins with an
    ignored <L>phrase<\L=PHRASES> is not to be tallied
    45    abort  accept  access  all  and  array  at  begin  body
    case  constant  declare  delta  digits  do  else  elsif  end
    entry  exception  exit  for  function  generic  goto  if  in  is
    limited  loop  mod  new  not  null  of  or  others  out  package
    pragma  private  procedure  raise  range  record  rem  renames
    return  reverse  select  separate  subtype  task  terminate  then
    type  use  when  while  with  xor    <L>Multi-tokens-file<\L=MULTI-TOKENS-FILE>
    47     If the option is present, then only the unique <L>phrases<\L=PHRASES> that
    are not wholly and everywhere contained in another <L>phrase<\L=ARGUMENT_LINE> are
    listed in the <L>tables of the output<\L=OUTPUT_FILE>
    48   In addition to the already specified <L>output<\L=OUTPUT_FILE>, if the option is
    present, then all the <L>sentences<\L=SENTENCES> are listed; if the option is
    present, then all the <L>tokens<\L=MULTI-TOKENS-FILE> are listed; if the option is present,
    then the <L>output<\L=OUTPUT_FILE> is verbose with the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords listed,
    and when the   m , and respectively the   x , option is present,
    the <L>multi-tokens<\L=MULTI-TOKENS-FILE>, and respectively the ignored <L>phrases<\L=PHRASES>, are listed
    49   If the option is present, then upper and lower case
    distinctions are to be applied in determining whether a <L>phrase<\L=PHRASES> is
    in a <L>sentence<\L=SENTENCES>
<\C>
<IE>CHUNK_FILE<\IE>
<H>(5) Chunk_file<\H>
<D>Chunk_file abstraction contains sentences about: free format (file).<\D>
<C>
     3       DESCRIPTION     All files mentioned in the synopsis
     provide their data in what is referred to as free format subject
     to particular restrictions to be described for each case
     4   In free format, the items of the file may be entered zero or
     several per line with a mixture of blanks and tabs before, in
```

```
     between, and after the items
     8     The  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  contains in free format a
     list of those <L>character strings<\L=STRING_TYPE_FILE> to be taken as
     <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords (see below)
    10   In each line, the <L>tokens<\L=MULTI-TOKENS-FILE> (see below) are in free format
    11   The optional  <L>multi-tokens-file<\L=MULTI-TOKENS-FILE>  contains in free format a
    list of those <L>character strings<\L=STRING_TYPE_FILE> consisting of more than one
    <L>symbolcharacter<\L=STRING_TYPE_FILE> (see below) which are to be taken as <L>multi-tokens<\L=MULTI-TOKENS-FILE>
    (see below)
<\C>
<IE>COVER<\IE>
<C>
<STL=COVER>FINDPHRASES ABSTRACTION NETWORK<\C>
<IE>INTRODUCTION<\IE>
<H>Introduction<\H>
<C>
This knowledge-base is a model of an abstraction oriented hypertext based on the findphrases case study.

Findphrase served as a case study becuase its decomposition is already known.

<NP>
The abstractions taken from Aguilera's program decomposition are:

Data Abstraction<TAB><TAB>Identifying Keywords

(1) <L>phrases<\L=PHRASES><TAB><TAB><TAB>phrase(s)
(2) <L>text_file<\L=TEXT_FILE><TAB><TAB><TAB>input, arbitrary text
(3) <L>argument_line<\L=ARGUMENT_LINE><TAB><TAB>argument, option
(4) <L>sentences<\L=SENTENCES><TAB><TAB><TAB>sentence(s)
(5) <L>chunk_file<\L=CHUNK_FILE><TAB><TAB><TAB>free format
(6) <L>multi_tokens_file<\L=MULTI-TOKENS-FILE><TAB>multi, token(s)
```

```
(7) <L>output_file<\L=OUTPUT_FILE><TAB><TAB>tables, output
(8) <L>string_type_file<\L=STRING_TYPE_FILE><TAB>strings, character(s)
(9) <L>punc_keyword_file<\L=PUNC_KEYWORD_FILE><TAB>punctuation keyword(s)

<NP>
Aguilera was acting the RA, and those data abstractions were found by Aguilera in implementing findphrases

The next page is some technical information on Hyperties, supplied automatically when generating a new hypertext form\
 the inside environment of Hyperties.

<NP>
This article, INTRODUCTION, can be used for an introductory article, or a table of contents.

The article called "<L>DESIGN INFORMATION<\L=]DESIGN INFORMATION>" includes information about how to use this design,\
 and the graphics files included with it.

You might also want to see the <L>Help System<\L=]HELP>, or explore the Search and Index screens by selecting the\
 buttons at the bottom of the screen.
<\C>
<SC>let $srchstr = "(none)"<\SC>
<IE>MULTI-TOKENS-FILE<\IE>
<H>(6) Multi_tokens_file<\H>
<D>Multi_tokens_file abstraction contains sentences about: multi tokens file<\D>
<C>
     2     SYNOPSIS  findphrases    number
     <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file    ignored-phrases-file
     multi-tokens-file
     7   If this <L>argument<\L=ARGUMENT_LINE> is not present, if it does not supply a
     number, or if the supplied number is outside the reasonable range
     of greater than zero and less than or equal to 50, then  number
     is taken as 10
```

```
 8     The  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  contains in <L>free format<\L=CHUNK_FILE> a
 list of those <L>character strings<\L=STRING_TYPE_FILE> to be taken as
 <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords (see below)
10   In each line, the tokens (see below) are in <L>free format<\L=CHUNK_FILE>
11   The optional  multi-tokens-file  contains in <L>free format<\L=CHUNK_FILE> a
list of those <L>character strings<\L=STRING_TYPE_FILE> consisting of more than one
<L>symbolcharacter<\L=STRING_TYPE_FILE> (see below) which are to be taken as multi-tokens
(see below)
14   These in turn are formed and classified into tokens and
<L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords based on the information provided by the
<L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  and, when the   m  option is present,
the  multi-tokens-file
21   whatever is in the  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file ; the
<L>symbolcharacter strings<\L=STRING_TYPE_FILE> are called <L>punctuation<\L=PUNC_KEYWORD_FILE> and the
wordcharacter <L>strings<\L=STRING_TYPE_FILE> are called keywords  Multi-token
22   whatever is in the  multi-tokens-file  Token
23   any word, <L>symbolcharacter<\L=STRING_TYPE_FILE>, or multi-token which is not listed
in the  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  <L>Sentence<\L=SENTENCES>
24   list of tokens delimited on each side by <L>punctuation<\L=PUNC_KEYWORD_FILE>/keyword
<L>Phrase<\L=PHRASES>
25   one or more consecutive tokens occurring within one <L>sentence<\L=SENTENCES>
27   The maximum length <L>phrase<\L=PHRASES> that has to be considered is that
of  number tokens
45    abort  accept  access  all  and  array  at  begin  body
case  constant  declare  delta  digits  do  else  elsif  end
entry  exception  exit  for  function  generic  goto  if  in  is
limited  loop  mod  new  not  null  of  or  others  out  package
pragma  private  procedure  raise  range  record  rem  renames
return  reverse  select  separate  subtype  task  terminate  then
type  use  when  while  with  xor    Multi-tokens-file
48   In addition to the already specified <L>output<\L=OUTPUT_FILE>, if the option is
```

```
    present, then all the <L>sentences<\L=SENTENCES> are listed; if the option is
    present, then all the tokens are listed; if the option is present,
    then the <L>output<\L=OUTPUT_FILE> is verbose with the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords listed,
    and when the   m , and respectively the   x , option is present,
    the multi-tokens, and respectively the ignored <L>phrases<\L=PHRASES>, are listed
<\C>
<IE>OUTPUT_FILE<\IE>
<H>(7) output_file<\H>
<D>Output_file abstraction contains sentences about: tables (of the) output<\D>
<C>
     7   If this <L>argument<\L=ARGUMENT_LINE> is not present, if it does not supply a
     number, or if the supplied number is outside the reasonable range
     of greater than zero and less than or equal to 50, then  number
     is taken as 10
    19   any printable <L>character<\L=STRING_TYPE_FILE> which is neither a wordcharacter nor a
    blank  Word
    30     The standard output consists of a copy of the <L>input<\L=TEXT_FILE> as is,
    with the lines numbered and the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords overstruck two
    times (i e , printed three times in place) so that they can be
    spotted easily, a frequency ranked table of the repeated <L>phrases<\L=PHRASES>
    31   i e , those appearing more than once among the <L>sentences<\L=SENTENCES>; that
    is the entries of the table are given in order of decreasing
    frequency, and an alphabetically ordered table of the repeated
    <L>phrases<\L=PHRASES>
    32     In the two tables, the entry for a repeated <L>phrase<\L=PHRASES> consists
    of
    34   In printing the repeated <L>phrase<\L=PHRASES> itself in a table entry, the
    underscores, , ''_'', are printed as blanks
    41   Only the repeated <L>phrases<\L=PHRASES> show up in the tables of the output
    47     If the option is present, then only the unique <L>phrases<\L=PHRASES> that
    are not wholly and everywhere contained in another <L>phrase<\L=PHRASES> are
```

```
    listed in the tables of the output
    48   In addition to the already specified output, if the option is
    present, then all the <L>sentences<\L=SENTENCES> are listed; if the option is
    present, then all the <L>tokens<\L=MULTI-TOKENS-FILE> are listed; if the option is present,
    then the output is verbose with the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords listed,
    and when the   m , and respectively the   x , option is present,
    the <L>multi-tokens<\L=MULTI-TOKENS-FILE>, and respectively the ignored <L>phrases<\L=PHRASES>, are listed
<\C>
<IE>PHRASES<\IE>
<H>(1) phrases<\H>
<D>Phrases abstraction contains sentences about: phrase(s), repeated phrases,
ignored phrases.<\D>
<C>
     1  findphrases   find repeated phrases in an arbitrary <L>text<\L=TEXT_FILE>
     2     SYNOPSIS  findphrases    number
     <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file    ignored-phrases-file
     <L>multi-tokens-file<\L=MULTI-TOKENS-FILE>
     6     The   n  <L>argument<\L=ARGUMENT_LINE> is optional and if present provides a
     number  number  serving as the maximum length phrase (to be
     described later) to be tallied
     9   The optional  ignored-phrases-file  contains one-per-line a
     list of phrases to be ignored in the tallying (see below)
    24   list of <L>tokens<\L=MULTI-TOKENS-FILE> delimited on each side by <L>punctuation<\L=PUNC_KEYWORD_FILE>/keyword
    Phrase
    26     The main job of this program is to tally the occurrence of
    all phrases in all <L>sentences<\L=SENTENCES>
    27   The maximum length phrase that has to be considered is that
    of  number <L>tokens<\L=MULTI-TOKENS-FILE>
    28   If the  ignored-phrases-file is provided, then the phrases
    given in the file are to be ignored in the tallying
    29   If the   b  option is used along with the
```

```
ignored-phrases-file, then phrases which begin with an ignored
phrase are also ignored in the tallying
30     The standard <L>output<\L=OUTPUT_FILE> consists of a copy of the <L>input<\L=TEXT_FILE---> as is,
with the lines numbered and the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords overstruck two
times (i e , printed three times in place) so that they can be
spotted easily, a frequency ranked <L>table<\L=OUTPUT_FILE> of the repeated phrases
31   i e , those appearing more than once among the <L>sentences<\L=SENTENCES>; that
is the entries of the <L>table<\L=OUTPUT_FILE> are given in order of decreasing
frequency, and an alphabetically ordered <L>table<\L=OUTPUT_FILE> of the repeated
phrases
32     In the two <L>tables<\L=OUTPUT_FILE>, the entry for a repeated phrase consists
of
33   a sequence of asterisks indicating the phrase's frequency as a
percentage of the maximum frequency; in this one asterisk
represents 10%, the actual number of occurrences of the repeated
phrase, the repeated phrase itself, and a list of the numbers of
all lines containing the beginning of the repeated phrase
34   In printing the repeated phrase itself in a <L>table<\L=OUTPUT_FILE> entry, the
underscores, , ''_'', are printed as blanks
36     Note that the definition of ''phrase'' is independent of the
number of times it occurs in the <L>sentences<\L=SENTENCES>
37   An  ignored phrase  is simply one to be ignored in the
tallying but not in searching for phrases
38   A phrase which contains an ignored phrase which itself is not
ignored is to be tallied
39   When the   b  option is present, a phrase which begins with an
ignored phrase is not to be tallied
40   A  repeated phrase  is one whose final tally is greater than
one
41   Only the repeated phrases show up in the <L>tables of the output<\L=OUTPUT_FILE>
42     Typically, the  ignored-phrases-file will contain so-called
```

```
    noise phrases such as ‘‘a’’, ‘‘an’’, ‘‘the’’, ‘‘of’’, ‘‘of the’’,
    etc plus any useless phrases found in previous runs of the program
    46                  This configuration is suited for finding
    repeated phrases in Ada   (Ada is a trademark of the U S Department
    of Defense) or in an Ada-based program design language
    47     If the option is present, then only the unique phrases that
    are not wholly and everywhere contained in another phrase are
    listed in the <L>tables of the output<\L=OUTPUT_FILE>
    48   In addition to the already specified <L>output<\L=OUTPUT_FILE>, if the option is
    present, then all the <L>sentences<\L=SENTENCES> are listed; if the option is
    present, then all the <L>tokens<\L=MULTI-TOKENS-FILE> are listed; if the option is present,
    then the <L>output<\L=OUTPUT_FILE> is verbose with the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords listed,
    and when the   m , and respectively the   x , option is present,
    the <L>multi-tokens<\L=MULTI-TOKENS-FILE>, and respectively the ignored phrases, are listed
    49   If the option is present, then upper and lower case
    distinctions are to be applied in determining whether a phrase is
    in a <L>sentence<\L=SENTENCES>
<\C>
<IE>PUNC_KEYWORD_FILE<\IE>
<H>(9) Punctuation_keyword_file<\H>
<D>Punctuation_keyword_file abstraction contains sentences about:punctuation keyword(s)<\D>
<C>
     2     SYNOPSIS  findphrases    number
     punctuation-keyword-file    ignored-phrases-file
     <L>multi-tokens-file<\L=MULTI-TOKENS-FILE>
     8     The  punctuation-keyword-file  contains in <L>free format<\L=CHUNK_FILE> a
     list of those <L>character strings<\L=STRING_TYPE_FILE> to be taken as
     punctuation/keywords (see below)
    14   These in turn are formed and classified into <L>tokens<\L=MULTI-TOKENS-FILE> and
    punctuation/keywords based on the information provided by the
    punctuation-keyword-file  and, when the   m  option is present,
```

```
    the  <L>multi-tokens-file<\L=MULTI-TOKENS-FILE>
    20   any sequence of wordcharacters delimited on each side by
    whitespace or a <L>symbolcharacter<\L=STRING_TYPE_FILE>  Punctuation/Keyword
    21   whatever is in the  punctuation-keyword-file ; the
    <L>symbolcharacter strings<\L=STRING_TYPE_FILE> are called punctuation and the
    wordcharacter <L>strings<\L=STRING_TYPE_FILE> are called keywords  <L>Multi-token<\L=MULTI-TOKENS-FILE>
    23   any word, <L>symbolcharacter<\L=STRING_TYPE_FILE>, or <L>multi-token<\L=MULTI-TOKENS-FILE> which is not listed
    in the  punctuation-keyword-file  <L>Sentence<\L=SENTENCES>
    24   list of <L>tokens<\L=MULTI-TOKENS-FILE> delimited on each side by punctuation/keyword
    <L>Phrase<\L=PHRASES>
    30     The standard <L>output<\L=OUTPUT_FILE> consists of a copy of the <L>input<\L=TEXT_FILE> as is,
    with the lines numbered and the punctuation/keywords overstruck two
    times (i e , printed three times in place) so that they can be
    spotted easily, a frequency ranked <L>table<\L=OUTPUT_FILE> of the repeated <L>phrases<\L=PHRASES>
    44    Punctuation-keyword-file
    45    abort  accept  access  all  and  array  at  begin  body
    case  constant  declare  delta  digits  do  else  elsif  end
    entry  exception  exit  for  function  generic  goto  if  in  is
    limited  loop  mod  new  not  null  of  or  others  out  package
    pragma  private  procedure  raise  range  record  rem  renames
    return  reverse  select  separate  subtype  task  terminate  then
    type  use  when  while  with  xor    <L>Multi-tokens-file<\L=MULTI-TOKENS-FILE>
    48   In addition to the already specified <L>output<\L=OUTPUT_FILE>, if the option is
    present, then all the <L>sentences<\L=SENTENCES> are listed; if the option is
    present, then all the <L>tokens<\L=MULTI-TOKENS-FILE> are listed; if the option is present,
    then the <L>output<\L=OUTPUT_FILE> is verbose with the punctuation/keywords listed,
    and when the   m , and respectively the   x , option is present,
    the <L>multi-tokens<\L=MULTI-TOKENS-FILE>, and respectively the ignored <L>phrases<\L=PHRASES>, are listed
<\C>
<IE>SENTENCES<\IE>
<H>(4) Sentences<\H>
```

```
<D>Sentences abstraction contains sentences about: sentence(s)<\D>
<C>
    23   any word, <L>symbolcharacter<\L=STRING_TYPE_FILE>, or <L>multi-token<\L=MULTI-TOKENS-FILE> which is not listed
    in the  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  Sentence
    25   one or more consecutive <L>tokens<\L=MULTI-TOKENS-FILE> occurring within one sentence
    26     The main job of this program is to tally the occurrence of
    all <L>phrases<\L=PHRASES> in all sentences
    31   i e , those appearing more than once among the sentences; that
    is the entries of the <L>table<\L=OUTPUT_FILE> are given in order of decreasing
    frequency, and an alphabetically ordered <L>table<\L=OUTPUT_FILE> of the repeated
    <L>phrases<\L=PHRASES>
    36     Note that the definition of ‘‘phrase’’ is independent of the
    number of times it occurs in the sentences
    48   In addition to the already specified <L>output<\L=OUTPUT_FILE>, if the option is
    present, then all the sentences are listed; if the option is
    present, then all the <L>tokens<\L=MULTI-TOKENS-FILE> are listed; if the option is present,
    then the <L>output<\L=OUTPUT_FILE> is verbose with the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords listed,
    and when the   m , and respectively the   x , option is present,
    the <L>multi-tokens<\L=MULTI-TOKENS-FILE>, and respectively the ignored <L>phrases<\L=PHRASES>, are listed
    49   If the option is present, then upper and lower case
    distinctions are to be applied in determining whether a <L>phrase<\L=PHRASES> is
    in a sentence
<\C>
<IE>STRING_TYPE_FILE<\IE>
<H>(8) String_type_file<\H>
<D>String_type_file abstraction contains sentences about: strings, characters<\D>
<C>
     8     The  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  contains in <L>free format<\L=CHUNK_FILE> a
     list of those character strings to be taken as
     <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords (see below)
    11   The optional  <L>multi-tokens-file<\L=MULTI-TOKENS-FILE>  contains in <L>free format<\L=CHUNK_FILE> a
```

```
    list of those character strings consisting of more than one
    symbolcharacter (see below) which are to be taken as <L>multi-tokens<\L=MULTI-TOKENS-FILE>
    (see below)
    13   The program parses the <L>text<\L=TEXT_FILE> into words and symbolcharacters
    17     blank , tab , newline ,   beginning-of-file , end-of-file
    Wordcharacter
    18     letter , digit ,  Symbolcharacter
    19   any printable character which is neither a wordcharacter nor a
    blank  Word
    20   any sequence of wordcharacters delimited on each side by
    whitespace or a symbolcharacter  <L>Punctuation<\L=PUNC_KEYWORD_FILE>/Keyword
    21   whatever is in the  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file ; the
    symbolcharacter strings are called <L>punctuation<\L=PUNC_KEYWORD_FILE> and the
    wordcharacter strings are called keywords  <L>Multi-token<\L=MULTI-TOKENS-FILE>
    23   any word, symbolcharacter, or <L>multi-token<\L=MULTI-TOKENS-FILE> which is not listed
    in the  <L>punctuation<\L=PUNC_KEYWORD_FILE>-keyword-file  <L>Sentence<\L=SENTENCES>
<\C>
<IE>TEXT_FILE<\IE>
<H>(2) Text_file<\H>
<D>Text_file abstraction contains sentences about: input, arbitrary text<\D>
<C>

     1  findphrases   find repeated <L>phrases<\L=PHRASES> in an arbitrary text
    12     No assumptions are made about the standard input, thus it
    may be an arbitrary text
    13   The program parses the text into words and <L>symbolcharacters<\L=STRING_TYPE_FILE>
    30     The standard <L>output<\L=OUTPUT_FILE> consists of a copy of the input as is,
    with the lines numbered and the <L>punctuation<\L=PUNC_KEYWORD_FILE>/keywords overstruck two
    times (i e , printed three times in place) so that they can be
    spotted easily, a frequency ranked <L>table<\L=OUTPUT_FILE> of the repeated <L>phrases<\L=PHRASES>
    36     Note that the definition of ‘‘phrase’’ is independent of the
```

```
    number of times it occurs in the sentences
<\C>
<IE>]ABOUT THIS DESIGN<\IE>
<H>ABOUT THIS DESIGN<\H>
<D>This knowledge base was created with Hyperties(r) Version 3.0 from Cognetics Corporation.  For more information\
 please contact:
Cognetics Corporation             609-799-5005
55 Princeton-Hightstown Road      800-229-TIES
Princeton Jct, NJ 08550           609-799-8555 (FAX)
<\D>
<IE>]DESIGN INFORMATION<\IE>
<H>ABOUT THIS DESIGN<\H>
<D>This article contains a description of some of the features of this knowledge base design.
<\D>
<C>
<RULER=10p,110p>This visual design is in EGA (640 by 350 by 16) graphics mode.

<LARGE>HELP SYSTEM<N>
A group of articles which make up a small help system are included in this knowledge base, and can be accessed from the\
 HELP button or with the F1 hotkey.  All of these articles have names beginning with "]HELP".

<LARGE>ARTICLES IN THE KNOWLEDGE BASE<N>
The article called "}LOOKUP ERROR" is displayed if a target article is missing when a link is selected.  It can be edited\
 to contain any text.

The article called "COVER" will be displayed first by the Browser.  If the "COVER" is missing, the article "INTRODUCTION"\
 will be displayed.

Variables used by this design are initialized in the Entry Script to the article "INTRODUCTION".

<LARGE>GRAPHICS NEEDED FOR THIS DESIGN<N>
```

There are several graphics files needed for this design. When the Author creates a new knowledge base using this design,\
 it also copies those files to the same directory. All of the graphics files have names that begin the short name for\
 the design (for example VGA2-DEF.PCX, EGA3-DEF.PCX).
<NP>
<LARGE>STYLESHEETS<N>
The stylesheets included with this design are:

<IN>DEFAULT <IN>This stylesheet will be used for most articles. No special command is needed to access this stylesheet.

<IN>COVER <IN>This stylesheet is used by the COVER article, and links to the INTRODUCTION article.

<IN>HELP <IN>This stylesheet is used by the Help System, and includes a limited set of buttons at the bottom of the screen.

<IN>INDEX <IN>This stylesheet is used only by the Index Screen.

<IN>HISTORY <IN>This stylesheet is used only by the History Screen.

<IN>SEARCH <IN>This stylesheet is used only by the Search Screen.

The text window in the DEFAULT stylesheet is 484 by 245 pixels, with the upper-left corner at 98, 37
<NP>
<LARGE>FONTS AND COLORS<N>
The article <L>]STYLE VIEW<\L=]STYLE VIEW> is used for viewing stylesheets, and contains a display of the fonts and\
 colors available in this design.

Here are the colors as they display in the text window. The numbers are used in the formatting <<COLOR=n> command, and\
 in the color specifications in the stylesheets.

<color=0>ZERO<color=1>ONE <color=2>TWO <color=3>THREE <color=4>FOUR <color=5>FIVE <color=6>SIX <color=7>SEVEN\
 <color=8>EIGHT <color=9>NINE <color=10>TEN <COLOR=11>ELEVEN <color=12>TWELVE <color=13>THIRTEEN\
 <color=14>FOURTEEN <color=15>FIFTEEN<color=*>

```
The palette for this design is:
<GRAPHIC=PALETTE,C>

<\C>
<IE>]EXPORTED TEXT<\IE>
<H>]EXPORTED TEXT<\H>
<C>
<TEXTFILE=EXPORT.TXT><\C>
<IE>]HELP<\IE>
<H>HELP<\H>
<C>
<stl=help><RULER=30p,60p,90p,120p>Help is available on the following topics

<TAB><L>Hot Keys<\L=]HELP: HOT KEYS>
<TAB><L>About Hyperties<\L=]HELP: ABOUT HYPERTEXT>
<TAB><L>Exporting and Printing Articles<\L=]HELP: EXPORT>
<\C>
<IE>]HELP: ABOUT HYPERTEXT<\IE>
<H>HELP: ABOUT HYPERTEXT<\H>
<C>
<stl=help>Hypertext is a special kind of computer database - or knowledge base -  designed for use in accessing text or\
 graphical material.

Hyperties(r), the hypertext software used to create this knowledge base uses an encyclopedia model in which a number of\
 articles on different topics are included in a book, and cross-references between topics allow a reader to delve more\
 deeply into a subject, or read supplementary material.

Like a published book, a hypertext knowledge base has both an author and readers.  Reading is also called\
 <L>browsing<\L=]HELP: BROWSING> and the program that displays the knowledge base is called the Browser.

As a knowledge base is created, the information is organised into small chunks called "articles' which are grouped\
```

```
 together into an electronic book.  Cross-references are included in the text of an article.  Text that is marked as a\
 reference to another article is called a <L>link<\L=]HELP: LINKS>.  Illustrations can also be included in an article.

In addition to the text of the article, there are commands on the screen which help a reader navigate through the\
 knowledge base.  These commands are called <L>buttons<\L=]HELP: BUTTONS> because you select them by putting the\
 cursor on them and "pressing" them with the ENTER key or a mouse click.  Many of the buttons also have <L>hot\
 keys<\L=]HELP: HOT KEYS> which can be used, to make browsing even easier.
<\C>
<IE>]HELP: BROWSING<\IE>
<H>HELP: BROWSING<\H>
<C>
<STL=HELP>The Browser is the program used to read a Hyperties knowledge base. In fact, you are using it right now!

To select a <L>link<\L=]HELP: LINKS> or a <L>button<\L=]HELP: BUTTONS>, use the mouse or cursor keys to move to the\
 highlighted term and press ENTER or click the mouse.

You can quickly move the cursor to any word by pressing the first letter or that word.  Many of the buttons also have\
 <L>hot keys<\L=]HELP: HOT KEYS> which can be used.
<\C>
<IE>]HELP: BUTTONS<\IE>
<H>]HELP: BUTTONS<\H>
<D>A button is a command on the screen.  Most buttons are "super links" which provide navigational short-cuts through\
 the knowledge base.<\D>
<IE>]HELP: EXPORT<\IE>
<H>HELP: PRINTING AND EXPORTING<\H>
<C>
<STL=HELP><RULER=36P>There are several hot-keys to help you export small blocks of text, export an entire article, or\
 send the text of an article to a printer.  These hot-keys are available in any article in this knowledge base.  If you\
 forget them, press F1 twice at any time to see a hot-key reference list.
```

TO PRINT AN ARTICLE:

<INDENT>Press F7. The entire content of the article you are reading will be sent to the printer attached to your LPT1\
 printer port. The content of the article is sent as ASCII text, with no printer control codes or formatting commands.\
  Short descriptions in popup boxes cannot be printed using this button.
<NEWPAGE>TO EXPORT AN ENTIRE ARTICLE TO A FILE:

<IN>Press F3. A dialog box will ask you to enter a name for the file to save the article in. The content text of the\
 article will be saved in this file.

<IN>If the file you name already exists, it will be replaced with the new file created using this button.
<NEWPAGE>TO COPY PART OF AN ARTICLE TO THE CLIPBOARD FILE:

<IN>Press F4. The selector bar will disappear, and be replaced by a cross-hair cursor. Mouse users will see the\
 pointer blink briefly.

<IN>Use the arrow keys or mouse to move the cursor to the upper-left corner of the text you want to clip. Press ENTER\
 or click the mouse.

<IN>Use the arrow keys or mouse to drag a box over the text you want to clip. When all of the text is enclosed in the\
 box, press ENTER or click the mouse again. The cursor will blink, and the selector bar will re-appear. The text\
 within the drag box is added to the end of a file named EXPORT.TXT on your default directory.
<NP>TO VIEW ALL OF THE TEXT YOU HAVE CLIPPED AND SAVED:

<IN>Press CTRL-F4. This will jump you to an article which displays the text you have clipped using the F4 function.\
  This text is already in a file on your local drive named EXPORT.TXT. You can use the F3 button to save it to a\
 different file name.

Graphics are not printed, or exported by Hyperties.
<\C>
<IE>]HELP: HISTORY<\IE>

```
<H>CTRL-H FUNCTION: DIRECTIONS FOR USE<\H>
<D>The History function lists the user's path of articles back to the beginning of the knowledge base.  The articles most\
 recently viewed appear on the top of the list.  All of these articles can be selected for re-viewing.

The History Screen can be accessed by selecting the History button from the Search Screen or Index Screen, or by\
 pressing F6 from any screen.
<\D>
<IE>]HELP: HOT KEYS<\IE>
<H>HELP: HOT KEYS<\H>
<C>
<stl=help><RULER=30p,140p,310p,410p>MOVING AROUND AN ARTICLE

<TAB>ESC<TAB>Return to Previous Article
<TAB>PgDn<TAB>Go to Next Page of Article
<TAB>PgUp<TAB>Go to Previous Page of Article
<TAB>HOME<TAB>Go to First Page of Article
<TAB>END<TAB>Go to Last Page of Article

HYPERTIES FEATURES

<TAB>HELP<TAB>F1<TAB><L>HISTORY<\L=]HELP: HISTORY><TAB>F6
<TAB><L>SEARCH<\L=]HELP: SEARCH><TAB>F2<TAB><L>PRINT<\L=]HELP: EXPORT><TAB>F7
<TAB><L>EXPORT<\L=]HELP: EXPORT><TAB>F3<TAB>LOG NOTES<TAB>F8
<TAB><L>CLIPBOARD<\L=]HELP: EXPORT><TAB>F4<TAB>RESTART<TAB>F9
<TAB><L>INDEX<\L=]HELP: INDEX><TAB>F5<TAB>QUIT<TAB>F10
<\C>
<IE>]HELP: INDEX<\IE>
<H>CTRL-F FUNCTION: DIRECTIONS FOR USE<\H>
<D>The Index is an alphabetical listing of all the articles in a knowledge base. Any article can be selected from the\
 list for browsing.
```

```
The Index Screen can be accessed by selecting the Index button, or pressing F5.<\D>
<IE>]HELP: INTRO<\IE>
<H>HELP: BROWSING<\H>
<D>Browsing - reading a Hyperties knowledge base is easy.  In fact, you are using it right now! Links (highlighted\
 phrases in the text) and buttons are used to move from article to article.  To select a link or button, click on it,\
 or move the selector bar to it and press ENTER.
<\D>
<IE>]HELP: LINKS<\IE>
<H>]HELP: LINKS<\H>
<D>A link is a word or phrase in the text which is marked as a reference to another article which contains additional\
 information on the link topic.<\D>
<IE>]HELP: SEARCH<\IE>
<H>HELP: SEARCH<\H>
<C>
<stl=HELP>Hyperties includes a full-text search facility.

You can search for all occurrences of a word or group of words within the knowledgebase.  This provides a quick way to\
 locate specific articles containing certain words or terms of interest.

To begin a search, select the Search button, or press F2, to go to the Search Screen.  On the Search Screen, select the\
 "SEARCH STRING" button and press ENTER.

You then enter a word to search for.  If you want to search for articles that contain more than one word you can join\
 them with an AND (or &) or OR (or |).  For example:

          meat and potato
          meat & potato

After you press ENTER to indicate that you are done entering the search string, a list of all articles containing the\
 word or words will be displayed.
```

```
You can then select any article from the list to read.  The articles which contain the search string most are at the\
 top of the list, so you can use this order to help you decide which article to read first.

The list is saved until you do another search and you can return to it at any time to select another article.  The list\
 is erased when you exit the knowledgebase.
<NP>
ADVANCED SEARCHING:

You can make a longer search string, for example:

    (beef or veal) and (potato or pasta)

to conduct a very specific search.
<\C>
<IE>]HELP: TEMPLATE<\IE>
<H>]HELP: TEMPLATE<\H>
<D>   F1     F2      F3     F4     F6       F7   F8     F9     F10
  HELP  SEARCH  EXPORT  CLIP  HISTORY  PRINT  LOG  RESTART  QUIT

 ESC - LAST ARTICLE;          PGDN/PGUP - NEXT/PREV PAGE;
 HOME/END - FIRST/LAST PAGE;  CTRL-F4 - VIEW EXPORT CLIPBOARD<\D>
<IE>]STYLE VIEW<\IE>
<H>DEFAULT<\H>
<C>
<STL=DEFAULT>This article is used to display a stylesheet with the VIEW command on the stylesheet index. The standard\
 font table is named STANDARD, and uses several fonts. You can add to the fonts now available, or customize the tables.\
 In addition to the normal font, the fonts now in the font table are:

<XS>EXTRA SMALL (Monospaced font HTBIT08R)
<IN>ABCDEFG abcdefg 1234567890 !@#$%^&*()
<SMALL>SMALL (Monospaced font HTBIT14N)
```

```
<IN>ABCDEFG abcdefg 1234567890 !@#$%^&*()
<LARGE>LARGE (Proportional font HTHEL20R)
<IN>ABCDEFG abcdefg 1234567890 !@#$%^&*()
<VL>VERY LARGE (Proportional font (HTHEL24R)
<IN>ABCDEFG abcdefg 1234567890 !@#$%^&*()<NORMAL>
<NP>The complete character set for the default and "normal" font (HTBIT14R) is:

  1- 10  <CHAR=1><CHAR=2><CHAR=3><CHAR=4><CHAR=5><CHAR=6><CHAR=7><CHAR=8><CHAR=9><CHAR=10>
 11- 20  <CHAR=11><CHAR=12><CHAR=13><CHAR=14><CHAR=15><CHAR=16><CHAR=17><CHAR=18><CHAR=19><CHAR=20>
 21- 30  <CHAR=21><CHAR=22><CHAR=23><CHAR=24><CHAR=25><CHAR=26><CHAR=27><CHAR=28><CHAR=29><CHAR=30>
 31- 40  <CHAR=31><CHAR=32><CHAR=33><CHAR=34><CHAR=35><CHAR=36><CHAR=37><CHAR=38><CHAR=39><CHAR=40
 41- 50  <CHAR=41><CHAR=42><CHAR=43><CHAR=44><CHAR=45><CHAR=46><CHAR=47><CHAR=48><CHAR=49><CHAR=50
 51- 60  <CHAR=51><CHAR=52><CHAR=53><CHAR=54><CHAR=55><CHAR=56><CHAR=57><CHAR=58><CHAR=59><CHAR=60>
 61- 64  <CHAR=61><CHAR=62><CHAR=63><CHAR=64>
 65- 90  ABCDEFGHIJKLMNOPQRTSUVWXYZ
 91- 96  [\]^_‘
 97-122  abcdefghijklmnopqrstuvwxyz
123-126  {|}~
128-170  ....
 !"#$%&’()*
171-179  +,-./012
219-223  [ ]^_
224-254  ‘abcde fghijklmno pqr}tuvwxy z{|}~
<NP><LARGE>Box Draw Characters<N>
IMKM; ZDBD? IMQM; ZDRD? ZDBD? UM8
LMNM9 CDED4 GDED6 FMNM5 FMXM5 TM>
: : : 3 3 3 : 3 : 3 : 3 @DADY VD7
HMJM< @DADY HMOM< @DPDY       SD=

<NP>You can display text in any color you want. Here are the colors as they display in the text window, and their\
 numbers:
```

```
<color=0>ZERO<color=1>ONE <color=2>TWO <color=3>THREE <color=4>FOUR <color=5>FIVE <color=6>SIX <color=7>SEVEN\
 <color=8>EIGHT <color=9>NINE <color=10>TEN <COLOR=11>ELEVEN <color=12>TWELVE <color=13>THIRTEEN\
 <color=14>FOURTEEN <color=15>FIFTEEN<color=*>

Links are displayed <L>like this<\L=]DESIGN INFORMATION> one, which links to an article with information about this\
 design.
<\C>
<IE>}LOOKUP ERROR<\IE>
<H>ZZZ LOOKUP ERROR<\H>
<D>This article is not here yet.<\D>
```