SODA: Software Designer's Aide

Jacob L. Cybulski

Amdahl Australian Intelligent Tools Program
Department of Computer Science and Computer Engineering
La Trobe University, Bundoora, Vic. 3083, Australia
Phone: +613 479 1270, Fax: +613 470 4915
Email: jacob@latcs1.lat.oz.au

1. Introduction

1.1 Background

1.1.1 **Aims.** Software Designer's Aide (SODA) is a computer-aided requirement acquisition system assisting software engineers in the construction and analysis of informal Software Requirement Specifications (SRS).

1.1.2 **Motivation.** SODA was conceived as a stand-alone SRS analysis tool. It is planned, however, that its interfaces will allow the tool integration with the AAITP HyperCASE environment, thus, permitting the informal SRS's to be entered via HyperEDIT editor, and the acquired information to be stored in HyperDICT, a data-dictionary based software repository. Consequently, SRS document contents will be accessible via authoring and navigation subsystems that HyperCASE provides.

1.1.3 **Methods.** The text-processing tools provided in SODA facilitate composition of requirement texts, their organisation into a template-based structure corresponding to one of the SRS standards, recognition of text references to reusable software components, inter-relating and cross-referencing of descriptions of data elements, behaviours and design criteria, finally SODA provides a re-use library that could aid the process of deriving software designs from informal specification documents and the subsequent hyper-navigation between them.

1.1.4 **Organisation.** SODA consists of three integral components :- the SRS compiler, its lexicon and grammar assisting the interpretation of text, and a set of program interfaces which allow the system to be integrated with

a text editor being the primary user interface and a data dictionary used to store references to re-usable software artefacts.

1.1.5 **Example.** SODA permits the user to first outline the document contents. The outline may conform to one of the internationally recognised standards which may be saved and reused when appropriate. In this example we decided on the format of UNIX on-line documents consistent with the template shown to the right. The template will be subsequently filled in with data to produce the actual document (below).

SRS Template
declaration
purpose
arguments
returned values
externals
method
constraints
pre-conditions
post-conditions
errors

SRS Text

declaration:
- accept(s, addr, addrlen)

purpose:
- function "accept" accepts a socket connection.

arguments:
- "s" is a socket to be accepted as part of a connection.
- "addr" is a pointer to the address of connecting entity "ce".
- "addrlen" is the length of address "addr" in bytes.

returned values:
- on error "accept" returns -1.
- on success "accept" returns a socket descriptor "ns".

externals:
- "q" is a queue of pending connections.
- "ce" is a connecting entity.

method:
- If "q" is empty then
  "accept" gets the first connection from "q"
  "accept" creates a new socket "ns" with the properties of "s"
  "accept" creates a file descriptor for "ns"
  else
  if "s" is marked as non-blocking
  then "accept" returns an error
  else "accept" blocks the caller until a connection is present.

constraints:
- "addr" format depends on the communication protocol.
- "accept" may not use "ns" to accept more connections.
- socket descriptor is a non-negative number.

pre-conditions:
- "s" was created with function "socket".
- "s" is of type SOCK_STREAM.
- "s" was bound to "address" with function "bind".
- "s" is listening for connections after a call to function "listen".
- "addr" points to the variable which will receive the address of "ce".
- "addrlen" contains the amount of space pointed to by "addr".

post-conditions:
- "addr" will return the address of the "ce".
- "s" will be left opened.
- "addrlen" will contain the length of the returned address.

errors:
- [EBADF] "s" is invalid.
- [ENOTSOCK] "s" is not a socket.
- [EOPNOTSUPP] "s" is not of type SOCK_STREAM.
- [EFAULT] Cannot write into "addr".

During the editing of SRS text, SODA checks the spelling, grammar and the style of typed phrases. When in doubt SODA will question the terminology and expressions used in the text, and will give the user an opportunity to rephrase the text to aid the clarity of the software description. The resulting document will predominantly consist of simple and unambiguous statements.

SRS Library
socket
file
connection
address
descriptor
queue
number
pointer
etc.

Once document editing is finished, SODA will analyse its contents with the aim to detect references to the programming notions which have been defined in this or other SRS texts (see right). Identification of such references allows tracing the use of re-usable concepts, checking the consistency of their use, indexing documents by contents, and the navigation between the concept and its original definition. SODA will also make explicit all the assumed or hidden information extracted from the body of the SRS and from the definition of referenced objects.

Examination of our sample specification by SODA reveals that "accept" is dependant on the definitions of functions mentioned in this spec, i.e. "socket", "bind" and "listen", but also that "accept" is related to the functions manipulating similar types of objects, i.e. "connect", "getsockname", "read", "recv", "select", "send", "shutdown", "socketpair" and "write". Semantic analysis of "accept" SRS may also help identifying inconsistencies in the terminology used across the document repository, e.g. "accept" uses the concept of "address" which is identical to the concept of a "name" used throughout the definition of "bind". SODA will advise the user of such inconsistencies.

Once the document is analysed and corrected, SODA will allow its browsing, searching, retrieval and navigation. Thus, the user will be able to display the names of all the functions mentioned or related to "accept", see the synonyms of "address", browse between all uses of "socket" in this and other documents, or to hyper-jump to the document defining the "socket" function.

In future SODA will also be able to analyse free-style and unstructured documents for the purpose of their indexing. This will aid adding existing software to SODA re-use library.

1.2 Purpose

The purpose of this document is to present the initial software requirements for the SODA system. It will satisfy the functional, performance, interface, design, and verification requirements of the system. This document is intended to be a baseline to supply sufficient information to the AAITP as a foundation for software design, assessment and approval. In the course of further research and prototyping activities, this document will be refined and expanded to constitute a formal software requirement specification.

1.3 Scope

The objective of this project is to describe the software requirements of the SODA system as encompassing the following deliverable products:

1.3.1 **Software Prototype**. The preliminary Prolog prototype of SODA will be operating only in a stand-alone mode. It will provide a SRS compiler capable of analysing requirement texts composed of very simple, unambiguous, and context free English statements, and storing the acquired information into a Prolog database. The prototype will also include a sample lexicon and grammar which could later be extended to suit any specific application domain.

1.3.2 **Software System.** The extendible software system operating under the HyperCASE environment integrating the SRS editor, the compiler, and the data dictionary, all allowing hypertext authoring and navigation between requirement and design documents.

1.3.3 **Research Report**. All research findings related to the concepts of SRS compilation, analysis, storage and subsequent hyper-navigation and processing will be collected in an extensive research report.

1.3.4 **Software Documentation**. Complete and easily understood documentation of the software will be provided to aid in future maintenance and/or modification of the software.

1.3.5 **User's Manual**. A user's guide is to be handed out to all SODA users. It will explain, step-by-step, exactly how to use the SODA and its main components.

1.4 Definitions, Acronyms, and Abbreviations

AAITP - Amdahl Australian Intelligent Tools Program, the joint venture between Amdahl Australia, La Trobe University and Prometheus Software Development, funded under the Victorian Government Offset Program, created to develop a range of prototypic UNIX-based CASE tools.

HyperBASE - The HyperCASE sub-system consisting of a number of tools organising and using a hypertext-based CASE repository.

HyperCASE - The hypertext-based CASE environment developed by AAITP, it consists of three major sub-components, i.e. the user interface (HyperEDIT), its knowledge base (HyperBASE) and data dictionary (HyperDICT).

HyperDICT - The HyperCASE data dictionary based software repository.

HyperEDIT - A customisable, object-oriented HyperCASE user interface.

IEEE SRS - Standard for writing requirement specification documents.

NLP - Natural Language Processing, a branch of Artificial Intelligence studying the methods of processing and representation of Linguistic knowledge.

PROLOG - A logics-based programming language.

SODA - Software Designer's Aide, one of the HyperBASE tools enabling the HyperCASE users to create and analyse informal software requirements.

SRS - Software Requirement Specification.

UNIX - Operating system being the SODA's delivery platform.

1.5 Document Overview

This document attempts to provide as much detail as possible to guide the tasks involved in the production of the SODA system, it includes the following three sections :-

Section 1 (Introduction) provides an overview of the entire SRS document;

Section 2 (General Characteristics) describes the product that will be produced, it includes: product perspective, product activity, user characteristics, general constraints, assumptions and dependencies;

Section 3 (Specific Requirements) addresses the specific requirements of the SODA system, it includes functional requirements, external interface requirements, performance requirements, design constraints, attributes, other requirements.

Appendix A (man accept) UNIX on-line document used as a basis of an example in this SRS.

Appendix B (SODA Template) sample document structure based on IEEE SRS standard.

Appendix C (SODA Lexicon) sample lexicon of frequently used technical terms to be specially treated by SODA.

2. General Characteristics

2.1 Introduction

This section introduces the software product. It describes the characteristics and limits affecting the product and its requirements.

2.2 Product Perspective

2.2.1 **Tool.** SODA will perform as a stand-alone tool to structure and to analyse the informal SRS documents. The resulting data dictionary, which is an extensive index to the SRS contents, may be futher processed by systems external to SODA to cross-reference software documentation, search for or navigate to the relevant documents, or check for documents completeness and consistency.

2.2.2 **Application.** Integration of SODA with HyperCASE will allow, HyperEDIT to act as its text editor and HyperDICT as a data dictionary. In such a configuration, SRS documents processed by SODA can later be used by HyperBASE sub-systems for the purpose of document refinement, hyper-navigation, or retrieval.

2.2.3 **Extensions.** By tailoring its lexicon and grammar, SODA can be customised to analyse any well structured technical documention written in clear and unambiguous English.

2.3 Product Functions

The SODA software will perform and facilitate the following functions:

2.3.1 **Structuring.** Provide the tool for defining and maintaining SRS standard templates (e.g. IEEE SRS) for easier structuring of SRS texts;

2.3.2 **Editing.** Organise an editing environment for entering the text of informal software requirements conforming to the selected SRS template, the form of English phrases accepted by the editor will be restricted to simple, unambiguous sentences, thus, promoting clarity of expression and facilitating automatic interpretation of user requirements;

2.3.3 **Validation.** Validate the user input for spelling, grammar, style and semantics, the changes to the form and contents of SRS text will be recommended when required (or possible), the validation and analysis rules may be customised to either suit a particular application or to improve the expressive power of the SODA grammar;

2.3.4 **Analysis.** Assist the user in identifying the reusable components mentioned in the text of SRS and relating them with references to other concepts in the same body of text;

2.3.5 **Compilation.** Present and verify with the user the cross-referenced collection of all concepts appearing in the body of SRS text, their interpretation and inter-relationships, such cross-references can be subsequently used for the purpose of concept refinement and hyper-navigation;

2.3.6 **Classification.** Classify and store the information extracted from the SRS text in the data-dictionary-based reuse library in a way promoting future text analyses, queries and retrieval of software artefact descriptions.

2.3.7 **Retrieval.** Facilitate traceability, validation and verification of SRS concepts in subsequent design tasks (when integrated with HyperCASE).

2.4 User Characteristics

We assume that SODA users will be those software engineers working closely with the clients of the specified system, and be responsible for writing informal, thus understood by the client, software requirements documents. The SODA users will be provided with all the relevant reference and user documentation, nevertheless, they will have to be trained in the use of the editor and be familiar with the restrictions imposed on its style and grammar.

2.5 Constraints, Assumptions and Dependencies

The following are general constraints for SODA:

2.5.1 **Platform.** The system is to be targeted for Amdahl/UTS, X Window System with OSF/Motif widget set, and relational DBMS, all the programs are to be written in C and Prolog programming languages;

2.5.2 **Environment.** The final version of the system will interact with the user via HyperEDIT user interface and will store its data in HyperDICT, both being integral parts of HyperCASE;

2.5.3 **Prototyping.** It is suggested that at the initial stages Apple Macintosh could provide a development platform with Prolog language as a prototyping environment, its database as SODA data dictionary, and HyperCard supplying the necessary text editing capabilities, the interfaces between the two tools could be organised in Think C.

3. Specific Requirements

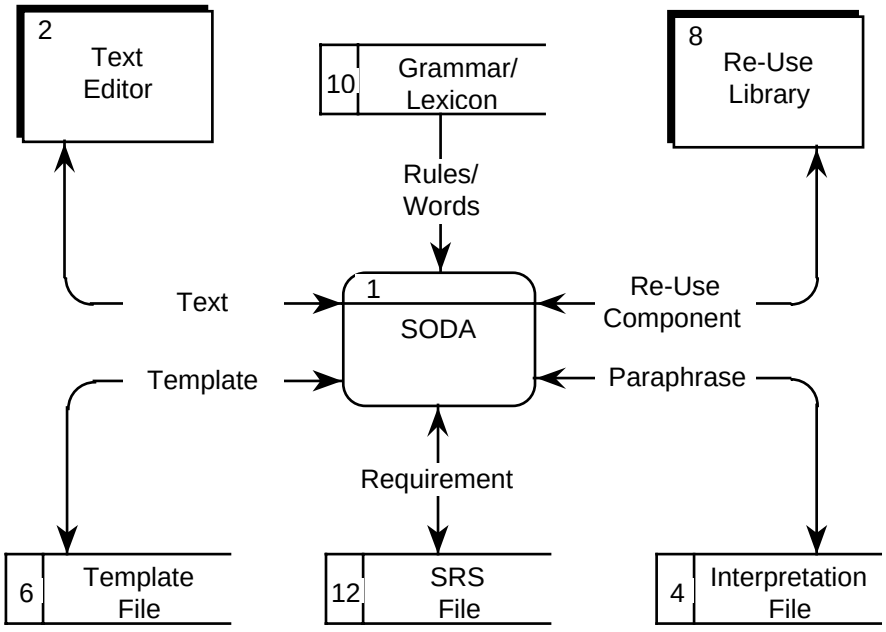
3.1 Functional Requirements

This section provides the details necessary for the systems engineer to create the design specification of SODA.

3.1.1 SODA Context

3.1.1.1 *Introduction*

SODA interfaces to the Text Editor responsible for the creation, editing and browsing the text of SRS template, requirement and interpretation documents. The Re-Use Library is used to store references to the re-usable components mentioned in the SRS text. The references and relationships between them are discovered by the SODA compiler relying on English grammar and lexicon of technical terms.



3.1.1.2 *Inputs*

- o Template and Requirement
- o Lexicon and Grammar

3.1.1.3 *Processing*

Extract references to re-usable software artefacts from informal SRS.

3.1.1.4 *Outputs*

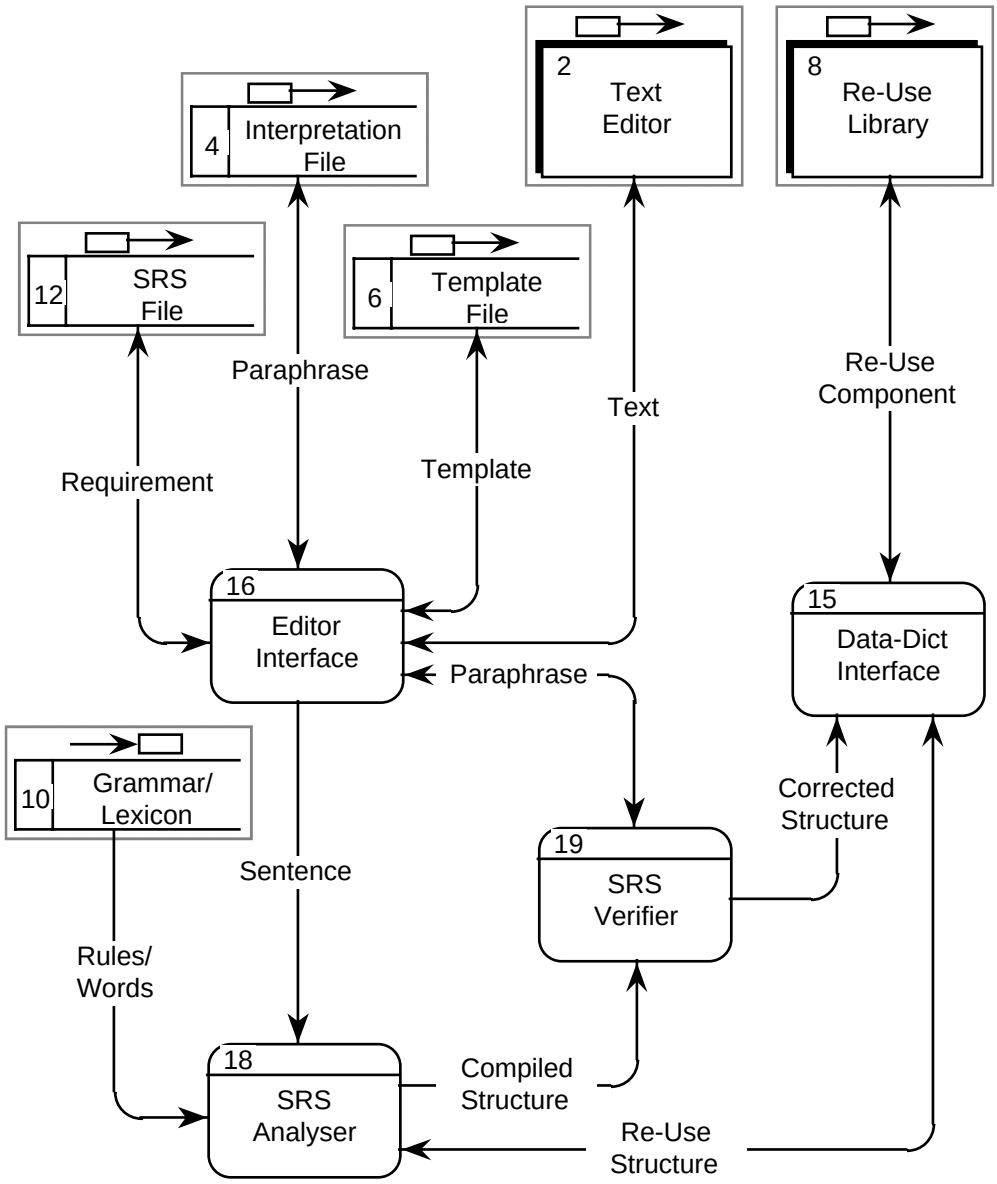
- o Re-Use components

- Interpretation file

3.1.2 SODA

3.1.2.1 *Introduction*

SODA allows the construction of structured SRS documents, which may be subsequently analysed in search of references to reusable software artefacts. After verification by the user SRS interpretation is stored in the library of re-usable software components. SODA provides programming interfaces to the generic text editor and a data dictionary.



3.1.2.2 *Inputs*

- Template and SRS Files
- Grammar and Lexicon

- Re-use Components

3.1.2.3 *Processing*

Use SRS template to create a SRS text
Analyse SRS sentences into compiled structures
Verify the correctness of compiled structures by paraphrasing them
Store the corrected structures into the re-use library

3.1.2.4 *Outputs*

- Interpretation file
- Re-use components

3.1.3 SRS Analyser

3.1.3.1 *Introduction*

SRS Analyser is used to interpret sentences of SRS text into their semantic representation. A lexicon and grammar are used to establish relationships between words appearing in the text, the semantic structures stored in the re-use library are used in resolving ambiguities. Pronoun references may have to be resolved by maintaining the sentence context for the duration of entire SRS section.

3.1.3.2 *Inputs*

- Sentence
- Lexicon and Grammar
- Re-use Structures

3.1.3.3 *Processing*

Use lexicon and grammar to produce sentence semantic structure
Unify the semantic structure with those of re-use components
Compile the final sentence structure

3.1.3.4 *Outputs*

- Compiled Structure (of the Sentence)

3.1.3 SRS Verifier

3.1.4.1 *Introduction*

The SRS Verifier takes the interpretation of SRS sentence and generates the sentence paraphrase which is subsequently verified by the user. This feedback is used to correct the sentence interpretation.

3.1.4.2 *Inputs*

- Compiled Structure (of the Sentence)

3.1.4.3 *Processing*

Use the compiled structure to generate the sentence paraphrase
Ask the user to verify the correctness of the paraphrase
Use the feedback to correct the structures

3.1.4.4 *Outputs*

- Paraphrase
- Corrected Structure

3.2 External Interface Requirements

3.2.1 User Interfaces

The user interface described in this section characterises the final submission of the SODA system. The initial prototype, though, will use a much more limited type of interaction.

3.2.1.1 **SODA Documents.** SODA produces and maintains three different types of SRS documents, i.e. the *template* defining the outline of the SRS document, then the document containing the *requirements*, and finally the document *interpretating* the SRS for the purpose of requirement verification by the user.

3.2.1.2 **SODA Editor.** The SODA system interacts with the user via a text editor allowing the creation, editing, analysis and browsing of SRS templates, texts and interpretations. The editor also provides the command interface allowing the invocation of SODA functions. The form of the user interaction with the editor is intentionally left unspecified, e.g. interactive or off-line, full-screen or line editing, single or multi-window displays, command-line, function and hot keys, static, pull-down or pop-up menus, can all be used.

3.2.1.4 **SODA Commands.** SODA operation normally starts with the construction or selection of an SRS template. The template will subsequently become the skeleton of the requirement document. The SRS interpretation is generated during the SRS analysis, and once created, the user may verify the requirements, accept them, reject them or rephrase the original requirement. All SODA functions are available

via groups of commands: filing, viewing, editing, browsing and formatting are available while manipulating any type of SRS document, outlining can be used to modify SRS templates, analysis is allowed on SRS texts, and verification on SRS interpretations only. The brief description of SODA commands follows.

3.2.1.4.1 ***File:*** *Allocating files to buffers*

New - create a new document
Open - open an existing document
Save - save newly created or modified document
Print - print the currently opened document

3.2.1.4.2 ***View:*** *Selection of the file buffer for editing or browsing*

Template - select the template buffer
Requirement - select the requirement buffer
Interpretation - select the interpretation buffer

3.2.1.4.3 ***Edit:*** *Modification of text in the current buffer*

Position - place the cursor in a specific text position
Select - select a continuous segment of text
Delete - deletes selected text
Cut - delete the selected text into a temporary storage
Copy - copy the selected text into a temporary storage
Paste - insert previously cut or copied text into text
Type - insert text by typing it
Undo - undo the last editing operation
Search - search for specific text or keyword
Replace - replace specific text or keyword
Next Section - go to the next template section
Repeat Section - repeat this template section

3.2.1.4.4 ***Browse:*** *Browsing the contents of the current buffer*

Char - character movement
Keyword - keyword browsing
Line - line scrolling
Sentence - sentence scrolling
Paragraph - paragraph scrolling
Page - page scrolling
Source - display a definition of the currently selected item

Reference - return from the item definition
Search - search for specific text or keyword

3.2.1.4.5 ***Format:*** *Changing the text visual attributes (as available)*
Character - change the character font, size and style (as available)
Paragraph - paragraph settings (as available)
Section - section settings (as available)
Document - documents settings (as available)

3.2.1.4.6 ***Outline:*** *Construct a document template*
Indent Heading - start the next level of section headings
Undent Heading - return to the previous level of headings
Next Heading - start the next heading of the same level
Comment Text - enter a template comment

3.2.1.4.7 ***Analyse:*** *Identification, classification and linking of text keywords*
Keyword - turn the selected text into a keyword
Classify - classify the selected keyword
Link - establish the relationship between selected keywords
Interpret - interpret the sentence into a set of linked keywords
Clear - clear all keywords of the selected text or sentence

3.2.1.4.8 ***Verify:*** *Verification of interpreted text*
Accept - accept current interpretation
Reject - reject current interpretation
Rephrase - rephrase current interpretation

3.2.2 Software Interfaces

3.2.2.1 **Editor Interface.** To become independent of the operating environment, SODA offers a programming interface to the virtual editor, functionality of which can be provided by several available programmable editors or systems supporting text editing, e.g. Emacs on UNIX, Alpha or HyperCard on Apple Macintosh, other editors may be custom linked with SODA via intelligent filters or pipes, e.g. Vi. Whatever editor is in use it must support the minimal set of commands available to the user and the SODA system:

3.2.2.1.1 ***File:*** *Allocating files to buffers*
New - create a new text document

Open - open an existing text document
Save - save newly created or modified text document
Print - print the currently document (possibly from the OS)

3.2.2.1.2 ***Edit:*** *Modification of text in the current buffer*

Position - place the cursor in a specific text position
Select - select a continuous segment of text
Delete - deletes selected text
Type - insert text by typing it
Insert - insert specific text string

3.2.2.1.3 ***Attribute:*** *Set text attributes*

Set Style - set the style of the current text item
Set Hilite - hilite the selected item
Set Index - set a text unique reference identifier

3.2.2.1.4 ***Search:*** *Search text*

Search Text - position the cursor at the next occurence of text
Search Index - position the cursor at the reference identifier

3.2.2.1.5 ***Status:*** *Enquire about the current context*

Get Cursor - find the current cursor position
Get Selection - find the address of the selected text
Get Text - get the text specified by text address

3.2.2.2 **Data Dictionary Interface.** The data dictionary is used to store references to SRS documents and relationships between such references. The system is assumed to be fully relational with access in some sort of embedded query language (e.g. E-SQL).

3.3 Performance Requirements

3.3.1 The response time for a single sentence compilation will be no more than five (5) seconds (slower in the initial prototype).

3.3.2 The time to construct the sentence paraphrase will be no more than five (5) seconds (slower in the initial prototype).

3.4 Design Constraints

3.4.1 Any editor meeting the minimum editing and browsing requirements can be interfaced to SODA.

3.4.2 Any relational database supporting embedded query language can be used as SODA re-use library.

3.4.3 The user will be given opportunity to verify SODA interpretation of requirements before committing it to its data dictionary.

A. man accept

NAME

accept - accept a connection on a socket

SYNOPSIS

```
#include <sys/types.h>
#include <sys/socket.h>

int accept(s,  addr, addrlen)
int s;
struct sockaddr *addr;
int *addrlen;
```

DESCRIPTION

accept accepts a connection on a socket. The argument s is a socket which has been created with socket(2), bound to an address with bind(2), and is listening for connections after a listen(2). accept extracts the first connection on the queue of pending connections, creates a new socket with the same properties of s and allocates a new file descriptor for the socket. If no pending connections are present on the queue, and the socket is not marked as non-blocking, accept blocks the caller until a connection is present. If the socket is marked non-blocking and no pending connections are present on the queue, accept returns an error as described below. The accepted socket, ns, may not be used to accept more connections. The original socket s remains open.

The argument addr is a result parameter which is filled in with the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the "communications domain" [see protocols(4)]. The addrlen is a value-result parameter; it should initially contain the amount of space pointed to by addr; on return it will contain the actual length (in bytes) of the address returned. This call is used with connection-based socket types, currently with SOCK_STREAM.

RETURN VALUE

The call returns -1 on error. If it succeeds it returns a non-negative integer which is a descriptor for the accepted socket (ns, described above).

ERRORS

The accept will fail if:

[EBADF]	The descriptor is invalid.
[ENOTSOCK]	Descriptor references a file, not a socket.

[EOPNOTSUPP]	The referenced socket is not of type SOCK_STREAM.
[EFAULT]	The addr parameter is not in a writable part of the user address space.

SEE ALSO

bind(2), connect(2), intro(2), listen(2), socket(2), intro(7).

B. SODA Templates

Many different document templates may be in use with the SODA system. It is proposed, however, that at least the IEEE standard will be fully supported. An example of such a template follows.

1. Introduction
    - 1.1 Background
    - 1.2 Purpose
    - 1.3 Scope
    - 1.4 Definitions, Acronyms, and Abbreviations
    - 1.5 Document Overview
2. General Characteristics
    - 2.2 Product Perspective
    - 2.3 Product Functions
    - 2.4 User Characteristics
    - 2.5 Assumptions and Dependencies
3. Specific Requirements
    - 3.1 Functional Requirements*
        - 3.1.1 Introduction
        - 3.1.2 Inputs
        - 3.1.3 Processing
        - 3.1.4 Outputs
    - 3.2 External Interface Requirements
        - 3.2.1 User Interfaces
        - 3.2.2 Hardware Interfaces
        - 3.2.3 Software Interfaces
        - 3.2.4 Communication Interfaces
    - 3.3 Performance Requirements
    - 3.4 Design Constraints
        - 3.4.1 Standards Compliance
        - 3.4.2 Hardware Limitations
    - 3.5 Attributes
        - 3.5.1 Security
        - 3.5.2 Safety
        - 3.5.3 Correctness
        - 3.5.4 Availability
        - 3.5.5 Reliability
        - 3.5.6 Efficiency
        - 3.5.7 Integrity
        - 3.5.8 Usability
        - 3.5.9 Maintability
        - 3.5.10 Testability
        - 3.5.11 Flexibility
        - 3.5.12 Portability
        - 3.5.13 Reusability
        - 3.5.14 Interoperability
        - 3.5.15 Other Factors

C. Software Terms

267	process
234	system
156	signal
116	directory
99	value
99	dump
89	argument
86	mode
83	error
79	string
79	data
78	function
68	char
66	specify
64	entry
61	stream
57	terminal
57	path
48	request
46	message
45	routine
45	program
45	block
44	indicate
43	trace
42	object
42	exist
41	space
41	modify
40	permission
40	image
40	flag
39	link
39	define
38	input
38	execute
35	screen
35	option
35	library
34	description
34	current
34	create
34	command
32	list
31	tape
31	status
31	operation
30	operate
29	occur
27	terminate
27	control
26	synopsis
26	conversion
25	network
25	array
24	environment
24	code
23	trap
23	host
23	field
22	zero
22	table
22	standard
22	ignore
21	structure
21	component
20	wait
20	float
19	interface
19	describe
19	correspond
18	update
18	search
18	integer
18	contain
17	remote
17	member
17	fault
17	effective
17	display
17	copy
17	access
16	length
15	volume
15	variable
15	unit
15	success
15	require
15	random
15	parent
15	location
15	interrupt
15	failure
15	cause
14	remain
14	receive
14	page
14	matrix
14	fail
14	exit
14	convert
14	buffer
13	shell
13	protocol
13	normally
13	extreme
13	deny
13	delete
13	associate
12	window
12	valid
12	size
12	level
12	empty
12	case
12	available
11	section
11	perform
11	index
11	format
11	fork
10	text
10	service
10	regular
10	range
10	match
10	map
10	log
10	exceed
10	element
10	deliver
10	condition
10	bug
10	apply
10	addition
9	stack
9	root
9	restore
9	operator
9	lock
9	encounter
9	disk
8	subsequent
8	store
8	retrieve

8 result
8 pipe
8 obtain
8 normal
8 maximum

SODA will use a lexicon of common English words expaded by a number of frequently used technical terms. Here is a sample of words frequently used in UNIX on-line documents.

*** Numbers denote the number of word occurencies found in a textsample.