1

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file - Votre référence

Our file - Notre référence

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30, and subsequent amendments.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30, et ses amendements subséquents.

Canada

**Multiactivity
as a
Restricted Object Based
Design Approach
by
Sylvain Lemire, B.C.S.S.**

**A Thesis Submitted to
the Faculty of Graduate Studies and Research
in partial Fulfilment of
the requirements for the degree of
Master of Science
Department of Systems and Computers Engineering**

Carleton University

Ottawa, Ontario

15, June, 1993

© copyright

1993, Sylvain Lemire

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

The author has granted an irrevocable non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of his/her thesis by any means and in any form or format, making this thesis available to interested persons.

L'auteur a accordé une licence irrévocable et non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de sa thèse de quelque manière et sous quelque forme que ce soit pour mettre des exemplaires de cette thèse à la disposition des personnes intéressées.

The author retains ownership of the copyright in his/her thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without his/her permission.

L'auteur conserve la propriété du droit d'auteur qui protège sa thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

ISBN 0-315-89865-8

Canada

Name Sylvain Lemire

Dissertation Abstracts International is arranged by broad, general subject categories. Please select the one subject which most nearly describes the content of your dissertation. Enter the corresponding four-digit code in the spaces provided.

Computer Science
SUBJECT TERM

0984
SUBJECT CODE

U·M·I

Subject Categories

THE HUMANITIES AND SOCIAL SCIENCES

COMMUNICATIONS AND THE ARTS

Architecture 0729
Art History 0377
Cinema 0920
Dance 0378
Fine Arts 0357
Information Science 0723
Journalism 0391
Library Science 0399
Mass Communications 0708
Music 0413
Speech Communication 0459
Theater 0465

EDUCATION

General 0515
Administration 0514
Adult and Continuing 0516
Agricultural 0517
Art 0273
Bilingual and Multicultural 0282
Business 0688
Community College 0275
Curriculum and Instruction 0727
Early Childhood 0518
Elementary 0524
Finance 0277
Guidance and Counseling 0519
Health 0680
Higher 0745
History of 0520
Home Economics 0278
Industrial 0521
Language and Literature 0279
Mathematics 0280
Music 0522
Philosophy of 0998
Physical 0523

Psychology 0525
Reading 0535
Religious 0527
Sciences 0714
Secondary 0533
Social Sciences 0534
Sociology of 0340
Special 0529
Teacher Training 0530
Technology 0710
Tests and Measurements 0288
Vocational 0747

LANGUAGE, LITERATURE AND LINGUISTICS

Language
 General 0679
 Ancient 0289
 Linguistics 0290
 Modern 0291
Literature
 General 0401
 Classical 0294
 Comparative 0295
 Medieval 0297
 Modern 0298
 African 0316
 American 0591
 Asian 0305
 Canadian (English) 0352
 Canadian (French) 0355
 English 0593
 Germanic 0311
 Latin American 0312
 Middle Eastern 0315
 Romance 0313
 Slavic and East European 0314

PHILOSOPHY, RELIGION AND THEOLOGY

Philosophy 0422
Religion
 General 0318
 Biblical Studies 0321
 Clergy 0319
 History of 0320
 Philosophy of 0322
Theology 0469

SOCIAL SCIENCES

American Studies 0323
Anthropology
 Archaeology 0324
 Cultural 0326
 Physical 0327
Business Administration
 General 0310
 Accounting 0272
 Banking 0770
 Management 0454
 Marketing 0338
Canadian Studies 0385
Economics
 General 0501
 Agricultural 0503
 Commerce-Business 0505
 Finance 0508
 History 0509
 Labor 0510
 Theory 0511
Folklore 0358
Geography 0366
Gerontology 0351
History
 General 0578

Ancient 0579
Medieval 0581
Modern 0582
Black 0328
African 0331
Asia, Australia and Oceania 0332
Canadian 0334
European 0335
Latin American 0336
Middle Eastern 0333
United States 0337
History of Science 0585
Law 0398
Political Science
 General 0615
 International Law and Relations 0616
 Public Administration 0617
Recreation 0814
Social Work 0452
Sociology
 General 0626
 Criminology and Penology 0627
 Demography 0938
 Ethnic and Racial Studies 0631
 Individual and Family Studies 0628
 Industrial and Labor Relations 0629
 Public and Social Welfare 0630
 Social Structure and Development 0700
 Theory and Method 0344
Transportation 0709
Urban and Regional Planning 0999
Women's Studies 0453

THE SCIENCES AND ENGINEERING

BIOLOGICAL SCIENCES

Agriculture
 General 0473
 Agronomy 0285
 Animal Culture and Nutrition 0475
 Animal Pathology 0476
 Food Science and Technology 0359
 Forestry and Wildlife 0478
 Plant Culture 0479
 Plant Pathology 0480
 Plant Physiology 0817
 Range Management 0777
 Wood Technology 0746

Biology

General 0306
Anatomy 0287
Biostatistics 0308
Botany 0309
Cell 0379
Ecology 0329
Entomology 0353
Genetics 0369
Immunology 0793
Microbiology 0410
Molecular 0307
Neuroscience 0317
Oceanography 0416
Physiology 0433
Radiation 0821
Veterinary Science 0778
Zoology 0472

Biophysics

General 0786
Medical 0760

EARTH SCIENCES

Biogeochemistry 0425
Geochemistry 0996

Geodesy 0370
Geology 0372
Geophysics 0373
Hydrology 0388
Mineralogy 0411
Paleobotany 0345
Paleoecology 0426
Paleontology 0418
Paleozoology 0985
Palynology 0427
Physical Geography 0368
Physical Oceanography 0415

HEALTH AND ENVIRONMENTAL SCIENCES

Environmental Sciences 0768
Health Sciences
 General 0566
 Audiology 0300
 Chemotherapy 0992
 Dentistry 0567
 Education 0350
 Hospital Management 0769
 Human Development 0758
 Immunology 0982
 Medicine and Surgery 0564
 Mental Health 0347
 Nursing 0569
 Nutrition 0570
 Obstetrics and Gynecology 0380
 Occupational Health and Therapy 0354
 Ophthalmology 0381
 Pathology 0571
 Pharmacology 0419
 Pharmacy 0572
 Physical Therapy 0382
 Public Health 0573
 Radiology 0574
 Recreation 0575

Speech Pathology 0460
Toxicology 0383
Home Economics 0386

PHYSICAL SCIENCES

Pure Sciences
 Chemistry
 General 0485
 Agricultural 0749
 Analytical 0486
 Biochemistry 0487
 Inorganic 0488
 Nuclear 0738
 Organic 0490
 Pharmaceutical 0491
 Physical 0494
 Polymer 0495
 Radiation 0754
 Mathematics 0405
 Physics
 General 0605
 Acoustics 0986
 Astronomy and Astrophysics 0606
 Atmospheric Science 0608
 Atomic 0748
 Electronics and Electricity 0607
 Elementary Particles and High Energy 0798
 Fluid and Plasma 0759
 Molecular 0609
 Nuclear 0610
 Optic 0752
 Radiation 0756
 Solid State 0611
 Statistics 0463
Applied Sciences
 Applied Mechanics 0346
 Computer Science 0984

Engineering
 General 0537
 Aerospace 0538
 Agricultural 0539
 Automotive 0540
 Biomedical 0541
 Chemical 0542
 Civil 0543
 Electronics and Electrical 0544
 Heat and Thermodynamics 0348
 Hydraulic 0545
 Industrial 0546
 Marine 0547
 Materials Science 0794
 Mechanical 0548
 Metallurgy 0743
 Mining 0551
 Nuclear 0552
 Packaging 0549
 Petroleum 0765
 Sanitary and Municipal 0554
 System Science 0790
Geotechnology 0428
Operations Research 0796
Plastics Technology 0795
Textile Technology 0994

PSYCHOLOGY

General 0621
Behavioral 0384
Clinical 0622
Developmental 0620
Experimental 0623
Industrial 0624
Personality 0625
Physiological 0989
Psychobiology 0349
Psychometrics 0632
Social 0451

Carleton University Memorandum

Date:

To: MICROMEDIA

From: Sneja Mouelhi

Re:

Please make a note that *Enclosure to thesis*
can be viewed on a diskette available only at Carleton
University Library.

The undersigned recommended to the Faculty of Graduate Studies
and Research acceptance of the thesis

'Multiactivity as a Restricted Object Based Design Approach'

submitted by

Sylvain Lemire, B.C.S.S.

in partial fulfilment of the requirements for
the degree of Master of Science

Thesis Supervisor

Thesis Supervisor

Chair, Department of Systems and Computer Engineering

Department of System and Computer Engineering

Faculty of Engineering

Carleton University

August 12, 1993

ABSTRACT

With the rapid development of micro-computers and the dropping cost of hardware, computers are becoming more and more available to everyone. Many companies started producing special software packages such as word processors, spreadsheets, desktop publishers, paint/graphic programs and database. To answer to the growing market demands. These "shrink-wrap" software packages demand a design methodology which can facilitate their rapid development and their numerous update releases with a constant user interface. The restricted object based design methodology presented in this thesis, focuses on the design of shrink-wrap software. The methodology advocates the separation of coordination from activity execution which is the base of the multiactivity paradigm. The methodology is implemented in three simple stages: the definition of event responses, the grouping of activities and the managers definition. This methodology takes advantage of the reusability of object based programming and the behavioral aspects of the multiactivity paradigm. Finally, the methodology is proven by providing an example system along with updates.

ACKNOWLEDGEMENT

The idea of applying the multiactivity paradigm to the design of shrink-wrap software came from previous research done by Dr. Moshe Krieger and his students. Dr. Krieger's valuable comments during our multiple discussions were a great help in the development of this new methodology and the preparation of this thesis. Thank You Dr. Krieger for all the 'good' time spent together, although sometimes very lively and hot, they were always extremely enlightening. Also I would like to thank Dr. Dorina Petriu for her work and advice.

And finally, I would like to thank Captain Ron Smith of the DAS ENG 5-4 organization at Canadian Forces Base Ottawa (Rockliffe) for reviewing and correcting my english syntax.

TABLE OF CONTENTS

	Page
Acceptance sheet	ii
Abstract	iii
Acknowledgement	iv
Table of Contents	v
List of Tables	x
List of Figures	xi

CHAPTER I: INTRODUCTION

1.1 SCOPE OF THE THESIS	1
1.1.1 Development of Shrink-Wrap Software	1
1.1.2 Software as a product	4
1.2 SOLUTION OUTLINE	9
1.3 THESIS OUTLINE	12

CHAPTER II: SOFTWARE SPECIFICATION

2.1 Software Development Process	14
2.1.1 Analysis Phase	15
2.1.2 Design Phase	16

2.1.3 Implementation and Testing Phases	17
2.2 Present Day Graphical Methods	17
2.2.1 Example System	18
2.2.2 Data Flow Graphs	18
2.2.3 Control Flow Graphs	22
2.2.4 Relation Diagrams	23
2.2.4.1 State Charts	25
2.2.4.2 Structure Diagrams	25
2.2.4.3 Object Based	28
2.3 PAD/PAL	29
2.3.1 Process Activity Diagram	30
2.3.2 Process Activity Language	32
2.3.3 Reactive System	36
2.4 Conclusion	36

CHAPTER III: THE RESTRICTED OBJECT BASED APPROACH

3.1 Overview	39
3.2 Design Procedure	41
3.2.1 Preliminary Design	41
3.2.1.1 Definition of Event Responses	42
3.2.1.2 Definition of Additional Constraints	42
3.2.1.3 Definition of the Data Structure	42

3.2.2 Grouping of Activities	43
3.2.3 Definition of Managers	43
3.2.3.1 Response Managers	44
3.2.3.2 System Manager	45
3.2.3.3 Event Manager	45
3.3 Implementation and Testing	46
3.4 Characteristic of Resulting System	46

CHAPTER IV: DESIGN EXAMPLE

4.1 Introduction	50
4.2 Problem Specification	50
4.2.1 General System Aspects	52
4.2.2 Commands Description	53
4.3 Design Specification	58
4.3.1 Preliminary Design	58
4.3.1.1 Event Responses	58
4.3.1.2 Additional Constraints	69
4.3.1.3 Data Structure Definition	70
4.3.2 Grouping of Activities	71
4.3.3 Definition of Managers	79
4.4 Comments on the Design	83
4.4.1 Debugging	86

4.4.1.1 Insert Appointment Bug	87
4.4.1.2 Appointment Comment Display	88
4.4.2 Testing	89

CHAPTER V: EXAMPLE OF SYSTEM UPDATE

5.1 Modification Overview	91
5.2 Password, Privacy and Messages	91
5.2.1 Additional Requirements	92
5.2.2 Update Procedure	94
5.2.2.1 Adding Passwords	94
5.2.2.2 Adding Privacy	99
5.2.2.3 Adding Message Recording	102
5.3 Time of Day Display	105
5.3.1 Update Procedure	105
5.4 Planning Option	106
5.4.1 Additional Requirements	107
5.4.2 Expected Changes	109
5.5 Comments	120

CHAPTER VI: CONCLUSION

6.1 Thesis Summary	122
6.2 Future Work	127

REFERENCES	130
-------------------	------------

APPENDIX A: Personal Agenda Program Listing	137
--	------------

LIST OF TABLES

TABLE	DESCRIPTION	PAGE
I	Personal Agenda System keyboard commands.	54
II	Text Box Editing key set.	56
III	Agenda System Activity List.	73
IV	Changes to Editing key set for message editing.	93
V	Modified execution modules.	96
VI	Modified execution modules for privacy update.	101
VII	Modified execution modules for message update.	104
VIII	Modified execution modules for planning update.	113
IX	Size of files of the personal agenda system.	126

LIST OF FIGURES

FIGURE	DESCRIPTION	PAGE
1.1	Object Based approach of multiactivity paradigm for shrink-wrap software design.	11
2.1	Simple package sorting system.	19
2.2	Data flow graphs primitives.	21
2.3	High level Petri net specification of the package sorting system.	24
2.4	Structured chart of the package sorting system.	26
2.5	Structured diagram of package sorting system.	27
2.6	Object Based representation of the package sorting system.	29
2.7	PAD ASN for the package sorting system.	33

2.8	Basic PAL control constructs.	34
2.9	PAL opcode of package sorting system.	35
2.10	Multiactivity coordinator layers.	37
3.1	Example of complete system diagram.	44
4.1	Agenda main windows layout.	51
4.2	Example window of Add, Delete or Modify command.	55
4.3	Example of a partial data structure.	72
4.4	Complete personal agenda system diagram (Part A).	84
4.5	Complete personal agenda system diagram (Part B).	85

CHAPTER I

INTRODUCTION

1.1 SCOPE OF THE THESIS

The purpose of this thesis is to suggest a new design methodology for the implementation of interactive multiversion software products, which are also known as shrink-wrap software.

1.1.1 Development of Shrink-Wrap Software

With the rapid development of the micro-computers and the dropping cost of hardware, computers are getting more and more available to everyone. Many software companies started producing special software packages such as word processors, spreadsheets, desktop publishers, paint/graphics programs, databases, as well as other applications, to answer the growing market demands. Originally, this software was designed specifically for each type of computer and without consideration for standard user interfaces. With the emergence of more and more naive user on the market, the push was on to provide more standardized products; this lead to today's shrink-wrap software.

The term "shrink-wrap software" became widely used, with various meaning, in popular computer literature around the 70's; a more precise definition of the term as found in [1] is: software that is mass produced to answer the needs of a group of users. This

means that users are buying software products off the shelf, products which are boxed and wrapped in shrunk cellophane. As stated in the same paper, the authors suggest that it is also quite appropriate for professional programmers to buy and use shrink-wrap software, "...programming is expensive and should be done only where the cost can be amortized across many usages. Consequently ... even the professional programmer, ..., are better off using a professionally polished commercial product than hacking up something."

Shrink-wrap software has a number of interesting characteristics:

- a. Companies manufacturing shrink-wrap software have to produce significant updates, at intervals of 6 to 24 months, in order to stay competitive by matching or bettering competitors options, and also to show their customers that they are still actively improving their products [2].
- b. With every new release of a shrink-wrap software product, the user interface must be as constant as possible, so that users do not have to retrain or question the improvements.
- c. New releases should be compatible with previous versions, so that the clients' existing data files (letters, graphics, forms, macros etc.) are not lost, or do not have to be re-defined.

d. New releases have to be fully tested before they hit the market. If a software product has too many bugs, clients lose faith in the company causing them to change to a competitor. Present design methods require a complete check of each new version which is very expensive and time consuming. Therefore it is desirable to use design methods that require only differential testing of new releases.

e. Another aspect of shrink-wrap software which has to be considered is that to be competitive it must be able to run on various computing platforms (different hardware and/or operating systems) [3].

f. By its nature the requirements of shrink-wrap software must satisfy a very large group of potential users. Since one cannot meet at once the requirements of all users, the product must be constantly updated; it must evolve over the years [2].

g. Probably the most important aspect of shrink-wrap software, as we are concerned, is that they are in effect man oriented event driven systems. These types of systems have two major attributes:

i) Because of the human interaction, these systems are basically sequential event driven systems (i.e. response time is not critical).

- ii) Each user action (event) corresponds to a specific set of operations that have to be executed in a given precedence relation to obtain the desired response.

The kind of shrink-wrap software products studied here are running on a very specific kind of computing platforms, namely mass market computers - (computers shipped in very large numbers) [1]. Today we have two major types of mass market computers for shrink-wrap software - personal computers (IBM PCs and clones of the 286 family and up, and Macintosh of the Mac II family), and workstations. Whether the software is running on a personal computer, which by nature is uniprocessor, or on a workstation, which may be a multiprocessor system, execution is still most often done in a sequential event mode. In cases where complex computation may be required or where response time may be too long, like in a 3D graphical software package, one could either use specialized processor and/or introduce some concurrency within each response.

1.1.2 Software as a product

One of the characteristics of shrink-wrap software as outlined above is that it is a product: which implies that it has to be engineered, thus implying that there exists a "Software Engineering" discipline. According to Mary Shaw, "...using the word 'engineering' to describe this activity takes considerable liberty with the common use of that term." [4].

The Webster's New World Dictionary defines engineering as follows:

Engineering: "The science concerned with putting scientific knowledge to practical uses" [5].

This is a very superficial definition of the word. Engineering implies well planned cost effective products. This involves identification of a need, the definition of the corresponding product, the design of this product, its implementation and its testing. From our point of view the most important aspect is that there exist methodologies and tools so that competent engineers can design "good" complex systems.

The problems of software engineering are exacerbated by the extremely large and growing variety of software products; software engineering problems can be categorized as:

a. Simple to complex. Software can be as simple as printing an address label or as complex as controlling a defense missile system. Many criteria can be used to distinguish between simple and complex software, such as: size of the program; data structures used; I/O performed; numerical processing performed, etc. It is important to realize that any single criterion does not make a program simple or complex, i.e., a small program is not essentially simple as it may be performing complex number crunching.

b. **One of a kind to large distribution.** Many software systems are developed for one particular application, others are mass produced for many possible customers.

c. **Fixed to variable:**

i) **adaptability:** is the program adaptable to a number of applications or specific to one problem.

ii) **number of versions:** is the program going to be produced a single time with no update or is it going to evolve with time, or be upgraded as needs change.

iii) **number of platforms:** is the software designed with a single computing platform in mind or is it trying to reach a larger market by being designed for an array of different platforms.

d. **Lifetime.** A software product with a very long lifetime must undergo a certain evolution to adapt to changing requirements or to readjust to changing hardware. Some embedded system software may have a life expectancy of up to 25 years.

- e. **Expert or naive user.** Software designed to be used by "ordinary" people with limited to no expertise versus software intended for computer specialists.
- f. **General or special purpose.** Special purpose software products are designed to address specific problems whereas general purpose deals with very large classes of problems.

There are probably many more attributes by which you can classify software. The point here is not to provide a comprehensive categorization but to indicate the large variety of present day software.

Yet today, in software engineering, designers are using ad hoc methods to develop products. As stated in [4], "We engineers begin by solving problems any way we can. After some time, we distinguish in those ad hoc solutions things that usually work and things that do not usually work. The ones that do work enter the folklore:... As the folklore becomes more and more systematic, we codify it as written heuristics and rules of procedure. Eventually, that codification becomes crisp enough to support models and theories, ... These can help improve practice, ..., the improvement in practice let us think about harder problems which we first solve ad hoc ... and so on."

In that same paper the author also identified five requirements for software design to become an engineering discipline:

- a. **"Understand the nature of expertise".** This relates to the required knowledge and skills needed by a software engineer to be able to practice his/her profession. This has to be addressed at the learning stages in schools and universities.
- b. **"Recognize different ways to get information".** This relates to how information is made available - retrieved from memory or references or derived anew each time. As stated in the reference, "...software engineering requires investment in the infrastructure cost - in creating the materials required to organize information." The cost of information retrieval depends also on whether it is done at the initial learning stage, used in practice or in an infrastructure.
- c. **"Encourage routine practice".** This relates to the fact that one needs consistent design methodologies and tools so that designers with ordinary talents can produce practical applications.
- d. **"Expect professional specializations".** Software engineering already recognizes the fact that different expertise are required for such problem area as real time systems, communications systems, language compilers, database management systems, etc. The main problem remaining still is that it is not widely accepted that we also need specialized design methodologies and tools to solve different problems.

- e. "Improve the coupling between science and commercial practice". Researchers must understand and address how to turn their results into useful elements of a production environment.

To a great extent we are at the very early stages of software engineering and still, unlike in many other engineering fields, many assume that one has to develop everything from scratch. In all other engineering disciplines it is recognized that a major role of an engineer is to integrate standard components in his design. This reusability factor is a major characteristic that led to the acceptance of object based programming.

1.2 SOLUTION OUTLINE

Today many software developers strongly advocate the use of object based methods. Because of having standard means of defining objects, these methods simplify the design procedures and improves reusability. This thesis proposes a restricted object based design methodology.

The problem with adopting a pure Object Based programming technique was that in event driven systems one needs to follow, with exactitude, the set of actions in a response; one needs to get into the objects. The other point in object-oriented, is that messages are transmitted freely between objects, this does not allow differential testing after a change; that is testing becomes a problem. The approach adopted here is a

restricted object based one which maintains reusability and ease of design found in object based programming. But by having a separate object manager which governs all communication between objects, it simplifies the tracing of responses and provides the possibility of differential testing after modifications to the system.

The methodology presented here addresses many of the engineering requirements identified by Mary Shaw in [4]. This new methodology encourages routine practice by defining specific design constraints. It recognizes reusability by creation of working elements. It improves coupling between science and commercial practices by providing a design methodology that fits a practical class of commercial products. It also looks at a specific area of software which could be considered as a professional specialization.

The restricted object based design methodology is founded on the multiactivity paradigm. Multiactivity system design is based on two observations. First, that all work has two components: the various activities and their coordination. Second, at implementation, if activity coordination and execution are separated, system design is simpler and more flexible [6]. In this scheme, shrink-wrap software is viewed as a set of responses, with each response corresponding to a temporal ordering of well defined operations by objects (see figure 1.1).

Functionally, multiactivity systems are implemented on three layers. On the top layer, the user interface and the system manager accept the environment's events and

Figure 1.1. Object Based approach of multiactivity paradigm for shrink-wrap software design.

reply by selecting the appropriate response manager. In the middle layer, the response managers coordinate the different responses. There is one response manager per event in the system, that is each event triggers one and only one response manager. In terms of object based, the first two layers are acting like an object manager. On the last layer activities are grouped into execution modules according to the resources they utilize. This is the layer in which the actual data processing is performed. Each module on this layer is like an instantiated object. Since these objects cannot communicate directly among each other they are regarded as restricted objects.

1.3 THESIS OUTLINE

Chapter II: Software Specifications; presents the software development process, followed by a discussion on the present day graphical methods. Then the Process Activity Diagram and Process Activity Language are introduced.

Chapter III: The Restricted Object Based Design Methodology; an overview of the methodology is presented first, then the designed procedure is explained in details and the characteristics of the resulting system are discussed.

Chapter IV: Design Example; provides an example system designed using the Restricted Object Based methodology. It starts with requirements specification of the example system from which the design specification is developed.

Chapter V: System Updates; this chapter discuss how updates to the system can be implemented.

Chapter VI: presents the conclusion of this thesis.

This thesis includes one appendix: The code listing for the final version of the example system.

This thesis also includes a high density diskette: Three directories on the diskette each containing the Turbo C code and a compile version of the example system; the first directory is for the original version as described in chapter IV, the second directory contains version 2.0 which includes the first two updates discussed in chapter V, and the third directory contains the final version (2.1) with all the updates discussed in chapter V.

CHAPTER II

SOFTWARE SPECIFICATION

2.1 Software Development Process

No matter what type of software development scheme that one considers, there are four basic steps encountered: Analysis, design, coding, and testing. Glenn Brookshear in his book "Computer Science an Overview" [7] makes allusion to the similarity of these steps and the phases of problem-solving presented by Polya in the late 1940s:[8]

- a. "Understand the problem";
- b. "Devise a plan for solving the problem";
- c. "Carry out the plan"; and,
- d. "Evaluate the solution for accuracy and for its potential as a tool for solving other problems".

There is nothing magic about this similarity since software development involves problem-solving. This chapter will look at the first two phases of software development with more emphasis on the design phase. Particular emphasis will be placed on the examination of the different present day design methodologies.

2.1.1 Analysis Phase

The main document produced in the analysis phase is the requirements specification. This document lists or identifies the requirements that a software application must satisfy in order to meet the needs of the user(s). Requirements are identified by the future user population or at the very least with their participation. The requirements specifications are basically the same regardless of the design methodology used. These requirements represent the user's needs not the program's needs. They should be written in a language that is familiar to both the users and the designers. They should be consistent and contain no contradictions from one part to the next. They should be complete, that is they should include all possible details and cover all conditions. And most importantly, they should be error free to eliminate any possible misinterpretations.

Unfortunately, with shrink-wrap software there exists no definite user organization. A software company producing shrink-wrap software must first probe the market needs for a certain type of product. After the needs for a product have been recognized, the potential users must be identified and possibly categorized. Only then can requirements be fully determined. The user, or potential user, participation in determining requirements is just as important here as with any other software. Requirements are gathered through surveys. The quality of these surveys, and the exactitude of their interpretation have direct consequences on the quality of the requirements specification, and ultimately on the success of the final product.

¹ This is another reason or characteristic of why shrink-wrap software has to be constantly updated (as discussed in section 1.1.1).

One interesting point brought up in [9], is that in addition to the requirements specification, this phase should include the draft of a user's manual. As the author says, "...the draft-copy user's manual forces the analyst (developer) to take a user's view of the software (particularly important with an interactive system)."

Of course there is a lot more about this phase than what was covered above. Some, like Zave [10], will argue that requirements specifications should be more structured or formalized. We rather see the formalization of requirements specification as the first step of the design phase.

2.1.2 Design Phase

The next step in software development is the design phase. The design phase consists of translating or interpreting the requirements specification into a more formal description of the proposed system. It is often divided into many stages. In early stages some modules are identified as black boxes. Throughout the stages the design specification is expanded until all the black boxes have been clarified.

The document produced here is a design specification, which roughly speaking corresponds to the partitioning of the requirements specification into implementable modules. The design specification should be done in a formal language which can be executed or simulated. This provides the designer, and the user, with a first level

² In many system's designs this is only a partial design specification, often one also has to specify the underlying computing platform (i.e. the system software and hardware).

validation of the proposed system. A good design specification should be well documented. This is very important when developing shrink-wrap software, it allows easier reference for future development and facilitates team design of a product.

2.1.3 Implementation and Testing Phases

The implementation phase corresponds to translating the design specifications into code. The testing phase is to check if the software works correctly and satisfies the requirements specification. This is a major problem in systems of any complexity.

2.2 Present Day Graphical Methods

Graphical design specifications are very convenient, not only are they formal (they can be interpreted) but they also provide a quick visual check for the designer. They also have the advantage that if they are simple enough, an application specialist can use them to produce a large part of the design specifications. If not he should be able to at least interpret them to conduct a functional check to see if the design specification truly reflects the requirements.

All graphical models are basically directed graphs with nodes representing various operations or modules and arcs that define interactions between nodes. In our view, there are essentially three types of graphical models: Data flow graphs, control flow graphs and relation diagrams. The first two types are precedence graphs indicating the data or control circulation within a system, while the third describes how the various modules of a

'call' on each other. It should be noted that our categorization does not always correspond with the nomenclature used by the different authors. It was observed that many authors will refer to their methodology by different names than used in our categorization. For example, in [11] Kiwamoto and Shigo refer to control flow when it would better fit our relation diagram definition.

2.2.1 Example System

At this point, before discussing the different types of graphical models, we introduce a simple system example which will be used to illustrate the various representations. Figure 2.1 represents a simple package sorting system. Packages arrive from the left on conveyer A, which moves them to point B. When a package reaches point B the conveyer is stopped until the robot picks it up. The robot takes a decision as to where the package goes (i.e. according to size, or weight, or color etc.) and moves it to conveyer 1, 2 or 3. Once in position the robot releases the package and returns to a ready position. The conveyer A and the robot's arm are free to execute concurrently whenever possible. It is assumed that conveyers 1, 2 and 3 are always moving, so we do not have to worry about them.

2.2.2 Data Flow Graphs

The data flow approach puts emphasis on data and the changes it undergoes as it moves through the system. The reasoning behind this approach is that data changes

Figure 2.1 Simple package sorting system.

should reveal the system's modules, and the activities each module must perform on data [7]. A good definition of data flow - "Sequence of data transformations from the initial input data set to the final output data set" can be found in [12].

At the highest level data flow graphs are composed of four main elements:

- a. Directed named arcs - representing the data flow.
- b. Circles - representing data transformation processes.
- c. Straight lines - representing files and databases.
- d. Boxes - representing external elements, sources and/or sinks.

At this level it is assumed that all decisions are made within the processing modules. To indicate the decision processes data flow graph must be augmented with extra primitives which provide data flow restrictions. Figure 2.2 presents some of the more common data flow elements.

Data flow oriented design methods are best suited to represent sequential 'transformation type' processing activities. Whenever one is considering concurrencies and/or reactive type processing, data flow diagrams do not provide complete system specifications. Therefore, some type of control specification needs to be added to data flow. This is what P. Ward did with his Transformation Schema [13]. He added control flow elements to the diagrams and Mealy-type state machines to describe the logic of controls.

Figure 2.2 Data flow graphs primitives

2.2.3 Control Flow Graphs

Control flow puts the emphasis on the controlled execution of a system. A good definition of control flow - "Arbitrarily ordered sequence of operations. End of the execution of one of them causes start of execution of another one" [12]. Control flow oriented design methods have been used more widely than data flow methods. Flow

charts are probably the most basic control flow diagrams. Flowcharts were invented to help programmers visualize assembly language programs [14], [20]. In flow charts the program control structures are emphasized while data structures are ignored.

Another well known graphical representation for control flow is a Petri network, or Petri net [15]. Although used more frequently to describe hardware, Petri nets have been applied to software. Petri nets includes of a set of places which can contain tokens, a set of transitions representing activities and a set of directed arcs connecting places to transitions and transitions to places. The state of the system, or net, is determined by the distribution of tokens at different places at a given point in time. Transitions are enabled when there is at least one token at each of its input places. When a transition is enabled it may be fired at which time one token is removed from each of the transition inputs and a token is added to each of its output places. Beyond a specification tool, various modifications of Petri nets (colored, timed, etc.) have become popular today to analyze

system behaviour. An example of the design specification of the package sorting system using Petri nets can be found at Figure 2 3.

There are also a number of specialized Petri nets. For example GRAFCET's (graphe de commande étape-transition) are modified Petri nets developed by researchers at the AFCET (Association Française pour la cybernétique Économique et technique) and the ADEPA (Association pour Le Développement de la Production Automatisé), for mechanical engineering and automatic production [16].

There are many more control flow representations, but we will look only at an additional two, PAD (Process Activity Diagrams) and PAL (Process Activity Language). PAD and PAL will be discussed in more details in the next section as they are closely related to the Restricted Object- Oriented design methodology presented in this thesis.

2.2.4 Relation Diagrams

In relation diagrams, often referred to as structure diagrams, a design is partitioned into modules that can call each other. The major drawback of relation diagrams is that they do not indicate the sequence of action, to do this one needs to introduce additional notation, like time threads [17]. There are, today, a number of relation diagrams. It seems that the first type were Structure Charts [18]. After this came Structure Diagrams which were developed to represent concurrent systems. And finally, Object Based system; they include a type of structured diagrams to represent interdependencies

Figure 2.3 High level Petri net specification of the package sorting system.

between various objects.

2.2.4.1 State Charts

State charts are a restricted type of relation diagrams. State charts are closely related to HIPO (Hierarchical, Input, Process, Output) charts which were developed by IBM [19]. They translate the 'functional picture'; a sequence of actions similar to data flow, into a hierarchical structure where higher modules can call lower modules. The lower modules are often numbered in the order they are called. A module is a set of one or more continuous program statements having a name by which other modules can invoke it. In state charts one should minimize connections between modules to minimize the paths along which changes and errors can be propagated to other parts of the system. Figure 2.4 gives the functional picture of the package sorting system example of section 2.2.1 and its corresponding state chart specification. In this example messages between modules are represented by the small arrows besides modules calls.

2.2.4.2 Structure Diagrams

Structure Diagrams generally reflect the software structure of the application programs, in this they represent partitioning of the system in functional modules and their interactions [14], [20]. There are a large variety of structure diagrams, here we introduce the Ada oriented structure diagrams as defined by Bowen [21] and Buhr [22]. Ada structure diagrams contain two kinds of modules: packages, or subprograms, represented by rectangles, and tasks represented by parallelograms. Arcs, or connecting arrows,

Figure 2.4 Structured chart of the package sorting system.

represent subprogram calls, entry calls, or data accesses from one module to another. These calls, or accesses, must be connected to entry points, which represent either an internal module subroutine or internal data storage. As in state charts messages are represented by small arrows with a bubble at the end. When more than one arc originates from a module the order in which they are serviced can be shown by numbering the arcs or by putting a curved arrow to indicate a top down, or bottom up order. Figure 2.5 shows the structured diagram for the package sorting system example of section 2.2.1.

Figure 2.5 Structured diagram of package sorting system.

2.2.4.3 Object Based

The basis for object based techniques is 'information hiding' [25][23]. Object based techniques partition a system into clusters of subroutines sharing "a common implementation or design secret from the rest of the software system" [25]. These clusters are objects. "An object has state, behaviour, and identity; the structure and behaviour of similar objects are defined in their common class; the terms instance and object are interchangeable" [23]. This means, an object can represent anything which may suffer the result of an action, like data structures, peripherals, subsystems, machines etc.. Most object based techniques can define three types of objects:

- a. Actor: An object that operates on others and is not operated on.
- b. Agent: An object that serves to perform some operations on behalf of other objects.
- c. Server: An object that is operated on and does not perform any operations on others.

In typical object based diagrams, at the higher level the objects are represented as clouds, their interdependencies are denoted by arcs and along these arcs messages transmitted are shown using small arrows. This seems to provide a very vague representation of a system, as you can see at Figure 2.6. This figure shows the object based view of the mail sorting robot example of section 2.2.1. To more suitably define a system one needs to represent an object's internal behaviour using finite state machines.

Figure 2.6 Object Based representation of the package sorting system.

2.3 PAD/PAL

The Process Activity Diagram (PAD) was developed at the University of Ottawa engineering department to provide a powerful, yet simple, tool for the specification and design of event driven multiprocessor systems [27]. Although complete and very easy to use, PADs present problems when used by inexperienced application specialists and design engineers. Improper use of certain constructs can lead to temporal errors, indeterminate behaviour caused by multiple triggers, or deadlocks. To resolve these problems the Process Activity Language (PAL), a structured form of PAD, was developed [28]. For these reasons the Restricted Object-Oriented design methodology presented in this thesis uses PAL.

2.3.1 Process Activity Diagram

Process Activity Diagrams are graphical representations of interdependencies between the activities of the responses processes of an event driven system. Each PAD describes a single event response. PAD's have two components: Activity Sequence Nets (ASNs) which are control flow diagrams representing sequencing relations, and Activity Timing Charts (ATCs) to indicate timing characteristics.

ASNs are composed of a number of nodes connected by directed arcs. Arcs represent internal and external triggers. Internal triggers are generated by the occurrence of some condition or by completion of an activity. External triggers are generated as a result of a condition outside the process. These conditions are normally identified by a label attached to the arc. There are two types of nodes, an activity initiator node (the only active nodes of the system) and a number of control nodes. The control nodes can be:

- a. **Branch nodes:** Enables an ASN to represent a selection of various activity sequences depending on some condition.
- b. **Merge nodes:** Provides a merging of the paths of a branch, the activation of only one of its input triggers activate all its output triggers. This is an OR type node.

- c. **Wait nodes:** The wait node allow joining of concurrent paths without executing an activity or doing anything else.
- d. **Gate nodes:** With the gate node one is able to block the propagation of a trigger under certain conditions. The gate node uses an enable and disable input to permit or block output trigger activation when input triggers are activated.
- e. **Timer/counter nodes:** The timer/counter node has only one input trigger. When enabled, an initial value, represented by the counter variable in the ellipse, is entered into the counter register, and every activation of its input trigger decrements the counter register. When the count reaches zero its output triggers are activated and the timer/counter node automatically disables itself.
- f. **End nodes:** Initially an ASN is considered dormant; it is awakened when its starting event occurs and it is put back to sleep, with all its nodes reset, when the end node is reached. This is the only node with no output triggers.

An ASN can have only one starting event, which may be latched, queued or volatile. It must also have an end node. In the case of processes with continued operations (unconditional feedback loop) the end node is reached only at system shut down. If needed, data flow between activities of a response can be indicated along the

trigger lines. Figure 2.7 depicts how the sorting robot example of section 2.2.1 can be implemented as a single PAD.

2.3.2 Process Activity Language

The Process Activity Language is a structured form of PAD and as such was developed to be a high-level language for the specification and design of event driven systems [28]. It allows designers to specify event response coordination without having to worry about their implementation details. In a sense we can say that PAL is a coordination language, like the one defined in [26]. Formalism and clarity are achieved in PAL by defining a system as well formed expressions using well defined, non complex, primitives which are similar to those found in structured programming languages [28].

"A system in PAL is specified as a set of processes $\{P_1, P_2, \dots, P_n\}$. A process P_i has a well defined starting event S_i , a response X_i , and an end E . The general form of a PAL process is represented by:

$$P_i : = S_i \bullet X_i \bullet E$$

"The body of a process i , the response X_i , is composed of PAL language primitives whose execution ordering is defined by PAL control construct" [28]. The different PAL constructs can be represented in algebraic form, in pseudocode, or in graphical form (see Figure 2.8). Within the construct only three language primitives are required to represent

Figure 2.7 PAD ASN for the package sorting system.

S_i = Package arrive at conveyer A

P_i = Process sorting robot

Concurrence

Sequence

```

    Wait Point B clear          ;signal from sensor
    wait Clear from robot      ;signal from robot
    start conveyer;
    wait at Point B            ;signal from sensor
    stop conveyer;
    wait for robot ready;      ;signal from robot
    Pick up package;
    set process var in robot claw;
    move package away;

```

end sequence;

sequence

CAS(Package in robot claw)

yes: CAS (sorting decision)

1: move robot to 1;

2: move robot to 2;

3: move robot to 3;

end cas

wait for in position ;signal from sensor

stop robot;

release package;

return to ready;

wait for robot ready ;signal from sensor

stop robot;

end cas

end sequence

end concurrence

End process.

Figure 2.8 PAL opcode of package sorting system.

a process; activity, set process variable and delay. Figure 2.9 shows the pseudocode for the package sorting system example of section 2.2.2.1.

Figure 2.9 Basic PAL control constructs.

2.3.3 Reactive System

Both, PAD and PAL, are aimed at reactive system design using the Multiactivity paradigm. According to this paradigm one can separate reactive systems into two parts, activity coordination and activity execution. "An activity is defined as a meaningful sequence of operations that can, once started, be executed to completion without having to wait for anything (information from other activities, resources, etc.)" [30]. In this terminology a process is the set of activities executed in a given precedence relation, to respond to a specific event.

The multiactivity coordination can functionally be organized as a three layer system (see Figure 2.10). The top layer, the system manager, monitors the environment for events, awakens or continues event responses and handles all interactions between event responses. The response managers layer is the next layer. It contains one manager for each active response which controls the sequencing of the response activities. The lower layer is the activity scheduler layer. It assigns activity executors to the ready to run activities.

2.4 Conclusion

According to [12], "Graphical representations correspond to 'programming at large': defining the 'processing modules' ('what' has to be done) and identifying 'how' they have to interact." In this context 'programming at small' corresponds to how the various modules should be implemented. The overall cost of a system, the design, the

Figure 2.10 Multiactivity coordinator layers.

implementation and the maintenance costs, is dependent on proper partitioning of the system [19]. The three types of graphical representation discussed earlier in this chapter, each have their own 'principles' on how partitioning should be done.

In data flows, partitioning is based on the data transformation. Data flows "represent sequences of transformations from an initial input data set to a final output data set" [12]. Emphasis is put on data flow while de-emphasizing the procedural aspect of the problem. Data flow graphs do not provide the ability to represent concurrencies. Because it is not known when data is available concurrencies between two modules is possible only when their data is independent. For these reasons the data flow partitioning

approach and graphical representations are better fitted for systems where large amounts of data manipulation is needed and none to little concurrency is possible.

In Control Flow, partitioning emphasizes the control structures of a system. In control flow graphical representations we do not only know how the controls are passed between modules, but we also know when these controls are available. This enables us to see parallelism in a system. Partitioning may not be as easy to determine at first, as with data flow, but it makes the control flow approach better suited for systems with concurrent ordered sequences of operations.

In relation diagrams, partitioning is based on a more functional decomposition of a system. Partitioning is done using module interrelations and their utilization of various resources. As we saw in section 2.2.4, there are different types of relation diagrams, each of which approaches partitioning in a slightly different way. In state charts partitioning is done using a functional picture of the system, while in structure diagrams partitioning generally reflects the software structure of the application programs, and in an object-oriented approach, partitioning is based around real and discrete objects. The overall relation diagrams partitioning approach and graphical representation shows that it is better suited for cases where resource manipulation and/or allocation is predominant.

CHAPTER III

THE RESTRICTED OBJECT BASED APPROACH

3.1 Overview

The Restricted Object Based approach is a modified form of the multiactivity design procedure. The multiactivity paradigm is based upon the separation of activity execution from coordination. Although the multiactivity design methodology was primarily aimed at real time systems implementation, the event driven nature of shrink wrap software makes multiactivity a sensible choice for its design. There are a few distinctions in how multiactivity is applied to real time system and shrink wrap software. Multiactivity applied to real time systems means that:

- a) Usually the executive runs on a separate processor and the activity executors on many different processors.
- b) System constraints are met by having a proper activity to processor allocation. To obtain the required availability a given activity may be included in a number of processors.
- c) Response execution corresponds to the proper scheduling of the component activities on the various processors.

In contrast, multiactivity applied to shrink wrap software means that:

- a) The executive and activity executors are all running on the same processor.
- b) Activities are grouped into execution modules according to selected attributes such as data requirements or resource utilization. Each activity can exist only in a single execution module.
- c) Response execution corresponds to the scheduling of the component activities via software calls to execution modules.
- d) These execution modules correspond to centrally coordinated objects. These 'objects' are passive in that they only respond to messages and do not initiate any messages on their own.

The Restricted Object Based methodology starts from the requirements specification. To allow for larger groups of participants to write the requirements specification, an informal approach is used; therefore it is not discussed any further here. In the following sections we will look at the design and implementation guidelines for the Restricted Object Based methodology.

3.2 Design Procedure

Designing a system using the Restricted Object Based methodology requires a design engineer to follow three simple stages: 1) preliminary design, 2) grouping of activities into execution modules, and 3) definition of the managers. Like in many other design methodologies these steps are carried out in an iterative fashion throughout a system's design.

3.2.1 Preliminary Design

Once the requirements of a system have been defined, the design engineer needs to interpret and formalize them. This formalization is required to allow verification and validation of the design specification by all parties involved in the system development process. Given the formalized requirements and that the design procedure does not introduce additional errors, it ensures that the implemented system will meet the requirements.

In the Restricted Object Based methodology, the Process Activity Language is used as the specification language. The preliminary design stage is in itself subdivided into three sub-stages: definition of event responses, definition of additional system constraints, and definition of the data structures.

3.2.1.1 Definition of Event Responses

As a first step, the requirements are translated into a set of event responses. Each event response is specified as the sequence of activities that has to be performed (i.e. that is each event response corresponds to a Process Activity Language process). In this sense the event responses correspond to the response managers of the system.

3.2.1.2 Definition of Additional Constraints

Often systems may not be purely defined as event responses, in these cases one needs to specify additional constraints. For example the special shape and colour of a particular window opened by a given command. The process for the command associated with this window would show that a special window is opened, but would not specify the exact shape and colour it should have; the shape and colour of the window are specified as constraints.

3.2.1.3 Definition of the Data Structure

For every system, one has to define a data structure of one kind or another. The proper choice of a data structure will vary depending on the various operations anticipated in the system, like sorting, searches, complex numbers operations, string manipulation, etc. It will also vary according to the programming language used to implement the system.

3.2.2 Grouping of Activities

From the event responses defined in the first stage, a list of required activities is made. By defining some attributes, like resources utilization, shared data, common structure operations, etc., one can divide the activities into categories. These categories are the execution modules of the system. These execution modules if properly constructed, have a similarity with objects in object based design in that they act as black boxes which execute the various activities when called upon to do so; they hide both the code and the data of the activities.

Once the grouping is done, the execution modules services must be defined. When a response manager, or any other manager, wants an activity to be executed, he makes a call to the appropriate execution module requesting a service. This service corresponds to the activity execution. This step may require a number of iterations. Activities may prove in some cases to be too large to leave as a single procedure. In such cases the activity should be broken down into smaller procedures to allow a better program modularity and readability. Alternately a number of activities which are always executed together can be replaced by a single larger activity.

3.2.3 Definition of Managers

There are three different types of managers which need to be defined: Response managers, a system manager and an event manager. Also in this stage a system diagram that shows the various interconnections between the managers and the execution modules

has to be provided. This diagram should reveal any design discrepancies which might have slipped by the designer, like inter-module requests which are illegal using the Restricted Object Based methodology. Figure 3.1 shows an example of such an interconnection diagram. As it can be seen there exists no connections between response managers or between execution modules.

Figure 3.1 Example of complete system diagram.

3.2.3.1 Response Managers

The event responses defined in the preliminary stage are now translated into response managers; translated in the sense that they are redefined in terms of execution module service calls.

To reduce the number of service calls to the various execution modules, consecutive calls to a given module can be regrouped into a single service. To further reduce service calls some low level processing, which may be required to set parameters etc., is executed in the response managers.

3.2.3.2 System manager

The system manager interacts either directly or indirectly via an event manager, with the environment and has a general knowledge of the system. In some cases the event manager can be included into the system manager as an event monitor implemented as an interrupt handler. In response to an event the system manager either initiate the appropriate response for this event or it passes the event to a response manager which was suspended by a 'wait for event' instruction. Associated with the system manager is a time, interaction and error handler which supervises the various timeouts and interactions between responses and initiates the proper responses to the various error (failure) conditions which may arise. It is possible that some applications use an error execution module, in such cases the error handler can be left out of the system manager. The overall picture of the system manager is that it looks like a table look-up for the various events of the environment.

3.2.3.3 Event Manager

The event manager is the link between the environment and the system; it is the user interface. The event manager collects events from the different environment

execution modules (i.e. modules which provide direct access to sensor devices), and passes them to the system manager. The event manager can be directed by the system manager to wait for the next event from any device or from a specific device (i.e. wait for the next keyboard input only).

3.3 Implementation and Testing

The implementation of the resulting system becomes very simple as it corresponds to coding a number of small modules. Translation of the managers is straight forward as it involves adding a few low level activities for the setting of parameters before some service calls. The coding of the execution modules corresponds to the coding of each activities.

The restricted object based design methodology leads to a systematize testing procedure. Each execution module can be tested separately by exercising each of the service calls for the various input conditions. Each response manager can be tested separately by building a condition table and checking the corresponding sequence of actions. For the modifications one only needs to test the new or modified execution modules or response managers.

3.4 Characteristic of Resulting System

The Restricted Object Based method provides a designer with a well defined procedure (strict guidelines) which encourages routine practices. It is a methodology

which, when understood and known by a competent engineer, allows him to design complex interactive software. These facts show that the Restricted Object Based design methodology is a step towards providing the requirements for software design to become an engineering discipline (see section 1.1.2).

Being that engineering disciplines must concern themselves with designing "good" complex systems at a 'reasonable' cost, we look at cost effectiveness of the Restricted Object Based design method. Yourdon and Constantine identified the following criterion to achieve minimum-cost systems [31]:

- a. "implementation cost" is minimized when parts of the problem are:
 - i. "manageably small," and
 - ii. "solvable separately."

- b. "maintenance cost" is minimized when parts of the system are:
 - i. "easily related to the application,"
 - ii. "manageably small," and
 - iii. "correctable separately."

- c. "cost of modifying a system" is minimized when the system's parts are:
 - i. "easily related to the problem," and
 - ii. "modifiable separately."

In the Restricted Object Based method, a system is partitioned into well defined modules which are easily related to the application, are relatively small and can be solvable and modifiable separately. Therefore meeting each of the cost effectiveness criterion.

The last point to consider is how well the Restricted Object Based methodology can be applied to shrink-wrap software. As indicated in Chapter 1 (section 1.1.1) shrink wrap software packages are man oriented event driven systems. The Restricted Object Based design methodology is aimed at such systems.

- a. It can provide a designer with a constant user interface throughout the evolution of a system.
- b. Because of the method's imposed design organisation and partitioning, systems developed using this method can be updated with relative ease.
- c. One can use differential testing for updates since they affect only specific parts of the system.
- d. Transportability of a system designed using this method can be assured by having to reprogram only the platform dependent system calls of the

various execution modules. The high level programming of the different managers assures their transportability.

As indicated by the above statements, the Restricted Object Based design methodology is a good, well founded software engineering design methodology.

CHAPTER IV

DESIGN EXAMPLE

4.1 Introduction

In this chapter we present a single user personal agenda system. This system is an automation of a desktop or pocket agenda in which a user can enter, modify, or delete an appointment entry in any day of any month. Where this system differs from a paper based agenda is that it covers more than one year.

Since the personal agenda system is introduced only as an example to indicate the applicability of the methodology, no effort was made to make it a sophisticated system neither was there any attempt to make the requirements specification fit the methodology.

In the following sections the requirements specification, the design specification using the Restricted Object Based method, and the final implementation of this personal agenda system are presented.

4.2 Problem Specification

We start by defining, in a point form, the required characteristics of the personal agenda system. These requirements are grouped in two categories, the general system aspects and the commands.

Figure 4.1 Agenda main windows layout.

4.2.1 General System Aspects

- a. The personal agenda has two main windows (see Figure 4.1). The first window, positioned on the left of the display area, is the month window. The second, positioned on the right of the display area, is the day window. In the month window the displayed month and year appear at the top, while in the day window the displayed day name and date are shown at the top.

At initialization the displayed month and day are set to the system's current date.

- b. The month window should be able to display any month of the present year, as well as at minimum any month of the past and future ten years (i.e. 1993 ± 10). Two buttons, <PREV> and <NEXT>, are provided to select the previous or next month.

The date of the displayed day is blinking in the month window. Every day of the current month with at least one appointment is displayed with the date characters in white on a lightgray background.

- c. The day window is able to display the schedule of any day of the displayed month. Selection of the displayed day is done using the mouse by clicking on the desired date in the month window.

The daily schedule is divided in half hour time slices; six of these slices are visible in the window at all times. An appointment may cover more than one time

slice, but only its first time slice can contain a comment. Scrolling buttons are provided to scroll the time slices up and down.

- d. For each operation: Add an appointment, Delete an appointment, Modify an appointment and Quit the personal agenda system, there is a button at the bottom of the day window. These buttons can be activated using the mouse.
- e. All operations and commands can be entered using either the mouse or the keyboard. The keyboard character set for operations and commands are shown in Table I.

4.2.2 Commands Description

- a. The three main operations, add, delete, and modify, only affect the current day. Each of these opens a window corresponding to a single appointment with a text box for the appointment start time, end time, and comment (see Figure 4.2). The name of the command executing appears at the top of the window. At the bottom there are the <OK> and the <Cancel> buttons. These buttons are activated by the mouse. Every command has a two minute inaction time limit. That is the operations cancels itself if no action from the user is detected after two minutes since the last action.

Key	Description
PgUp	Scroll up one month
PgDw	Scroll down one month
Rarr	(Right arrow) Current day plus one, stay within current month
Larr	(Left arrow) Current day minus one, stay within current month
Uarr	(Up arrow) Scroll schedule up
Darr	(Down arrow) Scroll schedule down
'A', 'a'	Add appointment
'D', 'd'	Delete appointment
'M', 'm'	Modify appointment
'Q', 'q'	Quit
<TAB>	(in add, delete, or modify) Move to next text box
<Ins>	(in add, delete, or modify) Select highlighted text box for data entry
<Enter>	(in add, delete, or modify) complete text entry or OK
<Esc>	(in add, delete, or modify) Cancel operation

Table I Personal Agenda System keyboard commands.

The text boxes can be activated using the mouse and edited using the key set shown in Table II.

The image shows a graphical user interface window titled "Add Appointment". The window is divided into several sections:

- Calendar:** A small calendar for January 1993 is located in the top left. It shows the days of the week (SUN, MON, TUE, WED, THU, FRI, SAT) and the dates.
- Date Selector:** A box on the top right displays "Sunday, January 10".
- Time Selector:** A list of times is shown on the right side, including "08:00", "10:00", and "10:30".
- Appointment Form:** A central form with the title "Add Appointment" contains three input fields: "Start", "End", and "Comment".
- Buttons:** At the bottom right, there are buttons labeled "<Cancel>" and "<OK>". At the bottom left, there are buttons labeled "<Add>", "", "<Mod>", and "<Quit>".

Figure 4.2 Example window of Add, Delete or Modify command.

Key	Description
<Home>	GoTo beginning of string
<End>	GoTo end of string
<Ins>	Switch between insert and overwrite
LArr	Move one character left
RArr	Move one character right
<BS>	Delete character on the left
	Delete character under cursor
<Enter>	Insert changes
<Esc>	Abort changes
All other	Enter character typed or ignore.

Table II Text Box Editing key set.

- b. When adding an appointment the start and end time fields cannot be blank when issuing the <OK>. If either one is blank the system reports an error, the appointment is not added and the Add operation is resumed. The comment field may be left blank. If a new appointment is conflicting with an existing appointment, the system reports an error and the new appointment is not added to the agenda. A <Cancel> aborts the add operation.
- c. To delete an appointment the user must first enter the start time or the end time, matching the appointment to be deleted. When either is entered, the system searches the displayed day for a match. If no match is found an error is reported

and the delete command restarted. If a match is found the appointment data is displayed in the appropriate text box. The appointment to be deleted is removed from the agenda only if an <OK> is issued at the end of the delete command. A <Cancel> aborts the delete operation.

- d. In the modify command the appointment to be modified is selected in the same manner as for the delete command. When a match is found the appointment's fields are displayed in the proper text boxes. Once the appointment to be modified is selected, an <OK> must be entered before modification can start. The user may select any field with the mouse and modify its contents using the editing key set of Table II. The changes are incorporated in the agenda only if the modify command is terminated with a second <OK> and there exist no time conflict with other appointments. If a time conflict exists, an error is reported, the modification is not carried out and the modify command is resumed. A <Cancel> command aborts the modify operation.
- e. The Quit command simply terminates the personal agenda system. All changes done during the system operation are saved for proper retrieval next time the agenda is started.

4.3 Design Specification

The design procedure is divided into three stages: preliminary design, activity grouping, and managers definition. These three stages for the personal agenda system are described in the following subsections.

4.3.1 Preliminary Design

Firstly, the event responses are defined using the Process Activity Language. Each event response (process) describes the sequence of activities required to execute the response to that event. Next the extra constraints imposed on the system are presented. And finally, the data structure for the personal agenda system is described.

4.3.1.1 Event Responses

By looking at the requirements one can see that the personal agenda has to respond to twelve events. These events and their associated response are listed bellow:

- a. **PREVIOUS_DAY:** Goto the previous day in the displayed month. This event is initiated by the left arrow key only.

```

Process Previous_day;
Sequence
Case
  (current_day > 1):
    current_day - 1;
    set_current_variables;
    fill_day_window;
    hide_cursor;
  End Case
End Sequence
END.
```

- b. **NEXT_DAY:** Goto the next day in the displayed month. This event is initiated by the right arrow key only.

```

Process Next_day;
  Sequence
    Case
      (current_day < month_size):
        current_day + 1;
        set_current_variables;
        fill_day_window;
        hide_cursor;
      End Case
    End Sequence
END.

```

- c. **SCROLL_DOWN:** Scroll down the time indexes displayed in the day window. This event is initiated by the down arrow key or by the scroll down mouse button in the day window.

```

Process Scroil_down;
  Sequence
    display_time + 30 minutes;
    fill_day_window;
    hide_cursor;
  End Sequence
END.

```

- d. **SCROLL_UP:** Scroll up the time indexes displayed in the day window. This event is initiated by the up arrow key or by the scroll up mouse button in the day window.

```

Process Scroll_up;
  Sequence

```

```

    display_time - 30 minutes;
    fill_day_window;
    hide_cursor;
  End Sequence
END.

```

- e. **SELECT_DAY**: Select a specific day, with the mouse, in the month window. This event is initiated with the mouse being clicked on any day square in the month window.

```

Process Select_day;
Sequence
Case
  (calendar_square in month):
    current_day := selected_day;
    set_current_variables;
    fill_day_window;
    hide_cursor;
  End Case
End Sequence
END.

```

- f. **MONTH_DOWN**: Change month displayed to previous month. This event is initiated either by the page down key or by the previous month mouse button in the month window.

```

Process Month_down;
Sequence
  current_month - 1;
Case
  (current_month == 0):
    current_month := 12;
    current_year - 1;
  End Case
  month_offset := month_first_day;
  fill_month_window;
  current_day := 1;
  set_current_variables;

```

```

    fill_day_window;
    set_appointments;
    hide_cursor;
End Sequence
END.

```

- g. **MONTH_UP**: Change month displayed to next month. This event is initiated either by the page up key or by the next month mouse button in the month window.

```

Process Month_UP;
Sequence
    current_month + 1;
Case
    (current_month == 13):
        current_month := 1;
        current_year + 1;
End Case
    month_offset := month_first_day;
    fill_month_window;
    current_day := 1;
    set_current_variables;
    fill_day_window;
    set_appointments;
    hide_cursor;
End Sequence
END.

```

- h. **ADD**: Add a new appointment to the displayed day. Add is initiated by the 'A' key (uppercase or lowercase) or by the ADD mouse button at the bottom of the day window.

```

Process Add_appointment;
Sequence
    open_add_window;
    hide_cursor;

```

```

RPT
wait_for_event;
Case
  (event = NEXT_BOX):
    current_box + 1;
    Case
      (current_box = 4):
        current_box := 1;
      End Case
    highlight_box;
    hide_cursor;
  (event = ACTIVATE):
    Case
      (current_box = 1):
        edit_start_time;
      (current_box = 2):
        edit_end_time;
      (current_box = 3):
        edit_comment_time;
    End Case
  (event = START_TIME):
    current_box := 1;
    highlight_box;
    edit_start_time;
  (event = END_TIME):
    current_box := 2;
    highlight_box;
    edit_end_time;
  (event = COMMENT):
    current_box := 3;
    highlight_box;
    edit_comment_time;
  (event = OK):
    Case
      ( (start_time not empty) and
        (end_time not empty) ):
        insert_appointment;
      ( either start_time or end_time is empty ):
        error_incomplete;
    End Case
  (event = CANCEL):
    Break out of RPT loop;
End Case
Until (event = OK) or (event = CANCEL);

```

```

        close_add_window;
        set_appointments;
        fill_day_window;
        hide_cursor;
    End Sequence
END.

```

- i. **DELETE:** Delete an appointment from the displayed day. The delete event is initiated by the 'D' key (uppercase or lowercase) or by the DEL mouse button at the bottom of the day window.

```

Process Delete_appointment;
Sequence
    open_delete_window;
    hide_cursor;
    RPT
        wait_for_event;
    Case
        (event = NEXT_BOX):
            current_box + 1;
            Case
                (current_box = 4):
                    current_box := 1;
            End Case
            highlight_box;
            hide_cursor;
        (event = ACTIVATE):
            Case
                (current_box = 1):
                    edit_start_time;
                    match_appointment_start_time;
                    Case
                        (appointment_matched):
                            fill_function_window;
                        (no_match_found):
                            error_no_match;
                            hide_cursor;
                    End Case
                (current_box = 2):
                    edit_end_time;
                    match_appointment_end_time;
                    Case

```



```

        (appointment_matched):
            fill_function_window;
        (no_match_found):
            error_no_match;
            hide_cursor;
        End Case
    End Case
    (event == START_TIME):
        current_box := 1;
        highlight_box;
        edit_start_time;
        match_appointment_start_time;
    Case
        (appointment_matched):
            fill_function_window;
        (no_match_found):
            error_no_match;
            hide_cursor;
    End Case
    (event == END_TIME):
        current_box := 2;
        highlight_box;
        edit_end_time;
        match_appointment_end_time;
    Case
        (appointment_matched):
            fill_function_window;
        (no_match_found):
            error_no_match;
            hide_cursor;
    End Case
    (event == OK):
        Case
            ( appointment_matched ):
                delete_appointment;
        End Case
    (event == CANCEL):
        Break out of RPT loop;
    End Case
Until (event == OK) or (event == CANCEL);
close_delete_window;
set_appointments;
fill_day_window;
hide_cursor;

```

End Sequence
END.

- j. **MODIFY:** modify an appointment in the displayed day. The modify event is initiated by the 'M' key (uppercase or lowercase) or by the MOD mouse button at the bottom of the day window.

```

Process Modify_appointment;
Sequence
  open_modify_window;
  hide_cursor;
  RPT # 1
    wait_for_event;
  Case
    (event = NEXT_BOX):
      current_box + 1;
  Case
    (current_box = 4):
      current_box := 1;
  End Case
  highlight_box;
  hide_cursor;
  (event = ACTIVATE):
  Case
    (current_box = 1):
      edit_start_time;
      match_appointment_start_time;
  Case
    (appointment_matched):
      fill_function_window;
    (no_match_found):
      error_no_match;
      hide_cursor;
  End Case
  (current_box = 2):
    edit_end_time;
    match_appointment_end_time;
  Case
    (appointment_matched):
      fill_function_window;

```

```

        (no_match_found):
            error_no_match;
            hide_cursor;
        End Case
    End Case
(event = START_TIME):
    current_box := 1;
    highlight_box;
    edit_start_time;
    match_appointment_start_time;
    Case
        (appointment_matched):
            fill_function_window;
        (no_match_found):
            error_no_match;
            hide_cursor;
    End Case
(event = END_TIME):
    current_box := 2;
    highlight_box;
    edit_end_time;
    match_appointment_end_time;
    Case
        (appointment_matched):
            fill_function_window;
        (no_match_found):
            error_no_match;
            hide_cursor;
    End Case
(event = OK):
    Case
        ( appointment_matched ):
            RPT # 2
            wait_for_event;
            Case
                (event = NEXT_BOX):
                    current_box + 1;
                    Case
                        (current_box = 4):
                            current_box := 1;
                    End Case
                highlight_box;
                hide_cursor;
            (event = ACTIVATE):

```

```

Case
  (current_box = 1):
    edit_start_time;
  (current_box = 2):
    edit_end_time;
  (current_box = 3):
    edit_comment;
End Case
(event = START_TIME):
  current_box := 1;
  highlight_box;
  edit_start_time;
(event = END_TIME):
  current_box := 2;
  highlight_box;
  edit_end_time;
(event = COMMENT):
  current_box := 3;
  highlight_box;
  edit_comment;
(event = OK):
  delete_old_appointment;
  insert_new_appointment;
Case
  (time_conflict):
    error_time_conflict;
    insert_old_appointment;
End Case
(event = CANCEL):
  Break out of RPT loop #2;
End Case
Until (event = OK) or
(event = CANCEL);
End Case
(event = CANCEL):
  Break out of RPT loop # 1;
End Case
Until (event = OK) or (event = CANCEL);
close_modify_window;
set_appointments;
fill_day_window;
hide_cursor;
End Sequence
END.

```

- k. **QUIT:** Quit the agenda system. The quit event is initiated by the 'Q' key (uppercase or lowercase) or by the QUIT mouse button at the bottom of the day window.

```

Process Quit;
Sequence
  clear_screen;
  save_agenda;
  close_agenda_file;
End Sequence
END.

```

- l. **INITIALIZE:** Initialize the agenda system. The initialize event is initiated by the start up of the agenda system.

```

Process Initialize;
Sequence
  init_agenda_windows;
  initcursor;
  current_month = system_month;
  current_day = system_day;
  current_year = system_year;
  current_time = system_time;
  display_time = round(current_time);
  month_offset = month_first_day;
  fill_month_window;
  open_agenda_file;
  retrieve_agenda;
  set_appointments;
  fill_day_window;
  hide_cursor;
End Sequence
END.

```

4.3.1.2 Additional Constraints

All windows in the personal agenda system have a blue background. Text and contour lines in the windows are in lightgray and the commands are in white. The required highlighted areas are in white on lightgray background. The different windows have the following coordinates:

a) month window: - top left corner: $x = 4, y = 2$

- bottom right corner: $x = 32, y = 18$

b) day window: - top left corner: $x = 37, y = 1$

- bottom right corner: $x = 76, y = 25$

c) function windows: - top left corner: $x = 5, y = 6$

- bottom right corner: $x = 75, y = 15$

NOTE: The function windows (add, delete and modify) are identical except for the title at the top.

The personal agenda system's data is retrieved from the file "AG##.\$\$\$". If the file is non-existent at the start of the system, it should be created automatically. On a "Quit" operation all data should be saved to that file.

4.3.1.3 Data Structure Definition

While executing, the system keeps the appointments information in the following data structure (see Figure 4.3):

```
typedef struct appt    /* appointment structure */
{
    char start_time[6];
    char end_time[6];
    char comment[36];
    struct appt *next; /* pointer to next appointment */
} *PAppt;
```

```
typedef struct days    /* day structure */
{
    char id[3];        /* date in characters */
    PAppt appointment; /* appointment for that day */
    struct days *next; /* pointer to next day */
} *PDays;
```

```
typedef struct months  /* month structure */
{

```

```

char mm[3];          /* month number */

PDays day_set;      /* list of days with appointment */
                    /* that month */

struct months *next; /* pointer to next month */

} *PMonths;

typedef struct years /* year structure */
{
    char iyyy[5];     /* year id in char */

    PMonths month_set; /* list of month with appointment */
                    /* that year */

    struct years *next; /* pointer to next year */

} *PYears;

```

4.3.2 Grouping of Activities

Using the event responses defined in section 4.3.1.1 we observe that there are thirty-six activities. These activities are listed in Table III.

To allow for better program modularity, the following activities were regrouped into a single activity. The parameter string to the new activity provides selection of the small differences that may exist between the activities regrouped.

Figure 4.3 Example of a partial data structure.

set_current_variables	match_appointment_start_time
fill_day_window	match_appointment_end_time
hide_cursor	fill_function_window
month_first_day	error_no_match
fill_month_window	error_notfound
set_apoointments	delete_appointment
open_add_window	delete_old_appointment
open_delete_window	insert_new_appointment
open_modify_window	insert_old_appointment
wait_for_event	clear_screen
highlight_box	open_agenda_file
edit_start_time	save_agenda
edit_end_time	retrieve_agenda
edit_comment	close_agenda_file
insert_appointment	system_month
close_add_window	system_year
close_delete_window	system_day
close_modify_window	system_time

Table III Agenda System Activity List.

open_add_window open_delete_window open_modify_window	open_function_window(type)
---	----------------------------

(type = {OPEN_ADD, OPEN_DEL, OPEN_MOD})

close_add_window close_delete_window close_modify_window	close_window
--	--------------

edit_start_time edit_end_time edit_comment	editstring(source)
---	----------------------

(= string top edit)

match_appointment_ start_time match_appointment_ end_time	match_appointment(select)
--	---------------------------

(select = {MATCH_ST, MATCH_ET})

delete_appointment delete_old_ appointment	delete_appointment(start)
--	---------------------------

(start = appointment to delete start time)

insert_appointment insert_new_ appointment insert_old_ appointment	insert_appointment(data)
--	----------------------------

(data = appointment to insert data)

system_month system_year system_day	system_date
--	-------------

open_agenda_file retrieve_agenda	retrieve_agenda
-------------------------------------	-----------------

close_agenda_file save_agenda	save_agenda
----------------------------------	-------------

error_no_match error_notfound error_incomplete error_time_conflict	error_em(error_id)
---	----------------------

(error_id = {NO_MATCH, NOTFOUND, INCOMPLETE, TIME_CONFLICT, AGENDA_MOD, WINDOW_MOD, NO_MOUSE, STRING_MOD, TIME_MOD, INITIALIZE_MOD})

Because of the simplicity of the error reporting activities, their regrouping is done directly into an execution modules. The additional error_id are for debugging purposes, they point to faulty modules.

To allow for a better program readability the following activities were broken down into smaller activities.

fill_day_window	fill_day_window fill_comment
-----------------	---------------------------------

set_appointments	set_appointments highlight_day
------------------	-----------------------------------

open_function_window	open_function_window open_window text_box
----------------------	---

editstring	editstring setcursor changecursor
------------	---

init_agenda_windows	init_agenda_windows command_line
---------------------	-------------------------------------

insert_appointment	insert_appointment new_year new_month new_day new_appointment
--------------------	---

retrieve_agenda	retrieve_agenda new_year new_month new_day new_appointment
-----------------	--

Note that the new_year, new_month, new_day, new_appointment activities in the insert_appointment break down and in the retrieve_agenda breakdown are the same.

By examining this list of activities one can regroup the activities into five categories (execution modules). The modules and the services they provide (activities they execute) are listed bellow, note that the activities are written as 'C code function and procedure headers. The services provided by the execution modules are written in uppercase.

a. Window execution module:

```

INITIALIZE
    void init_agenda_windows(void);
    void command_line(int x, int y);
OPEN_ADD
OPEN_DELETE
OPEN_MODIFY
    void open_function_window(int typ);
    void save_screen(int x1, int y1, int x2,
                    int y2, char *buffer);

```

```

void open_window(int x1, int y1, int x2,
                 int y2, int bkgnd, int frame, char *title);
void text_box(int x, int y, int length, char *title);
COORD_TO_EVENT
int coordinate_to_event(char *parameter);
FILLMONTH
void fill_month_window(void);
FILLDAY
void fill_day_window(void);
void fill_comment(char *bottom);
FILLCOM
void fill_function_window(void);
SETAPP
void set_appointment_in_month(void);
void highlight_day(int id);
HIGHLIGHT
void highlight_box(int boxnum, int mode);
CLOSE
void close_window(void);

```

b. Agenda execution module:

```

INSERT
RETRIEVE
OPEN
PYears new_year(void);
PMonths new_month(void);
PDays new_day(void);
PAppt new_appointment(void);
int insert_appointment(char *parameters);
void retrieve_agenda(void);
SAVE
CLOSE
void save_agenda(void);
ADELETE
int delete_appointment(char *parameters);
SETVAR
int set_current_variable(void);
MATCH_ST
MATCH_ET
int match_appointment(int match_select,
                      char *param);

```

c. Error execution module:

```

INITIALIZE_MOD
    error_in_initialize_module;
TIME_MOD
    error_in_time_module;
WINDOW_MOD
    error_in_window_module;
AGENDA_MOD
    error_in_agenda_module;
STRING_MOD
    error_in_string_module;
NO_MOUSE
    warning_no_mouse_detected;

NO_MATCH
    warning_no_match_found;

```

d. String execution module:

```

INITIALIZE
    void initcursor(void);
EDIT
    void setcursor(unsigned int shape);
    void changecursor(int insmode);
    int editstring(char *s, int posx, int posy,
                                                           int maxlength);
HIDE
    void hidecursor(void);

```

e. Date/time execution module:

```

JULIAN
FIRST_DAY
    long julday(int mm, int id, int iyyy);
    int mod(long x, int y);
CALDAT
    void caldat(long julian, int *mm, int *id,
                                                           int *iyyy);
TIME
    void system_time(char *time_now);
DATE
    void system_date(char *today);

```

```
COUNTER
    void timer_value(char *val);
```

Further down in the design process we find that two more execution modules are required to regroup the keyboard and mouse activities. The extra two modules are as follows:

a. Mouse execution module:

```
INITIALIZE
    int mouse_installed(void);
    void show_mouse_cursor(void);
    void set_mouse_cursor(int Ordchar );
SYSOFF
    void hide_mouse_cursor(void);
TRY
    void move_mouse(int xcoord, int ycoord);
    void get_mouse_action(int *but, int *hor,
                                                                    int *ver);
    int mouse_pressed(int button);
    int mouse_released(int button);
```

b. Keyboard execution module:

```
NEED
    int getch();
TRY
    int kbhit();
KEY_TO_EVENT
    int change_key_to_event;
```

4.3.3 Definition of Managers

First we need to define the event response managers. This is done simply by rewriting the event response processes defined in section 4.3.1.1 and replacing the activity calls by the execution module service calls defined in section 4.3.2. Because this is a

simple translation they are not repeated here; the final code, including updates of chapter V, can be found in appendix A.

The next step is to define the system manager. This manager controls the responses of the system. Every response manager that requires some kind of event (character) input must request it through the system manager. The system manager Process Activity Language process is as follows:

```

Process system_manager;
Sequence
Case
    (service = EDITING):
        return(get_text_character);

    (service = WTE_inst):
        stamp := get_counter_value;
        get_event;
        Case
            (NOEVENT):
                elapse = get_counter_value;
                Case
                    ((elapse - stamp) >= 120):
                        return(CANCEL);
                End Case;
                return(event);
        End Case;

    (service = MOUSED):
        initialize_mouse;

    (service = SYSTEM):
        initialize_manager;
        initialize_mouse;
        RPT
            get_event;
        Case

```

```
(event = ADD):  
    add_manager;  
  
(event = DELETE):  
    delete_manager;  
  
(event = MODIFY):  
    modify_manager;  
  
(event = MONTH_UP):  
    month_up_manager;  
  
(event = MONTH_DOWN):  
    month_down_manager;  
  
(event = SELECT_DAY):  
    select_day_manager;  
  
(event = SCROLL_UP):  
    scroll_up_manager;  
  
(event = SCROLL_DOWN):  
    scroll_down_manager;  
  
(event = NEXT_DAY):  
    next_day_manager;  
  
(event = PREVIOUS_DAY):  
    previous_day_manager;  
  
(event = QUIT):  
    quit_manager;  
    terminate := 1;  
  
    End Case;  
    Until( terminate = 1);  
    End Case;  
    shutdown_system;  
End Sequence;  
END.
```

2

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

The last step is to define the event manager or user interface. Here we can notice that extra activities are required. These activities are specialized input device actions (i.e. mouse and keyboard input activities, like `get_mouse_action`, `mousePressed`, `mouseReleased`, `getch`, `keypressed` etc.). Note that these activities were included in section 4.3.2. The Process Activity Language event process is as follows:

```

Process user_interface;
Sequence
Case

  (service == TEXT):
    ch := getch;
    return(ch);

  (service == EVENT):
    Case
      (keyboard_hit):
        ch := getch;
        event = key_to_event;
        return(event);
      (default):
        return(0);
    End Case;
  Case

    (ch == 0):
      ch := get_mouse_action;
      Case

        (ch == MLeftB ):
          event := coordinate_to_event;
          return(event);

          (default):
            return(NOEVENT);
        End Case;
      End Case;
  End Case;

```

```

(service == INITIAL):
    initialize_mouse;

(service == SHUTDOWN):
    turn_off_mouse;
End Case;
End Sequence;
END.

```

The final part of the design procedure is to provide a complete system diagram. This diagram appears at Figure 4.4 and Figure 4.5.

4.4 Comments on the Design

In this section, the implementation of the personal agenda system is discussed. The comments will be restricted to how the application of the Restricted Object Based design methodology affected the system design and implementation processes. Some areas of this example system could have been programmed in a more efficient way, but it must be kept in mind that it is only an "example".

Design and implementation of the coordinating part of the system is fairly simple. By following the design procedure laid out in the method, one can define and code the managers with no trouble. The hardest thing is that one must remember that activity execution is not done in the coordination layer.

The activity restrictions on the executors, or execution modules, proved to be the hardest part to follow in implementation, in that one is often tempted to bypass

Figure 4.5 Complete personal agenda system diagram (Part B).

coordination to acquire data from another execution module. For example, when editing a string it seems more natural to acquire characters directly from the keyboard doing a 'getch' function call instead of having to request the character from the system manager, or doing a call to the date/time execution module to get the date instead of having the response manager do the call and then provide the date to the new module. But these restrictions turned out to be very useful in understanding and debugging the overall system.

4.4.1 Debugging

Following the rules proved to be a great help in comprehension and debugging. The example system is a reasonable size program which could involve more than one programmer for its implementation. The Restricted Object Based design methodology permits each programmer to understand the complete system. This is very important when it comes to debugging any system. In this example, debugging was particularly easy. Most of the debugging follows normal debugging practices used in any other programming techniques, that is using breakpoints, when available, or print message to pin-point to the troubled area. Where it is easier, is that having a program structured this way, the area with a bug is limited to one response manager and one execution module.

Furthermore, the response manager is so simple that it usually does not contain any bug. The following paragraphs relate two debugging examples.

4.4.1.1 Insert Appointment Bug

During testing, it was noticed that two appointments could not be entered in the same day and that no time conflict error was reported when the second appointment was entered in the same time period as the first. These errors pointed to the agenda execution module and the add response manager. By looking at the add response manager, it was found that it had no fault; every call to execution modules was done properly. The only other option was that the fault had to be in the insert activity of the agenda execution module.

The insert activity was one of the first activities to be implemented. It was fairly large and complex. Before even considering to trace this fault, it was noted that about half of this procedure involved locating the year, month and day where the new appointment was to be inserted. Furthermore it was observed that the modify and the delete activities also had to locate the year, month, and day in their execution. These last two activities did this by using the sub-activity "set current variables" which sets the global pointer to the current-year, current-month and current-day if they exist in the structure or to null if they do not. Therefore, half of the insert activity was eliminated and the same sub-activity was used to identify the location of the appointment. The only problem area left to be checked was whether the new appointment was inserted properly. It was found that the comparison of the existing appointment times against the new appointment time was incorrect. The comparison check was corrected and the end of the appointment list was simplified to fix the bug.

This bug, although somewhat major, required only one hour to locate, fix and test. Note that not only was the bug fixed, but the program was also optimized.

4.4.1.2 Appointment Comment Display

An appointment reserved time is shown by highlighting the reserved time slots in the day window. When an appointment was scheduled in the middle of the day the time slots display was working correctly. When an appointment was scheduled at the end of the day or very early in the day, say around 01:00 in the morning, the time slots display was erratic. This bug pointed directly to the fill_comment activity in the window execution module.

The fill_comment activity assigns a number to each time slots in a day, starting at 00:00 hrs, number 0, to 23:30 hrs, number 47. This activity assumed that the numbers of the displayed slots would always have the smallest number as the top of the display and the largest number as the bottom of the display (i.e. the display is showing time from 15:00 hrs to 17:30 hrs, the top slot number is 30 and the bottom is 35). Therefore it only checked if any part of an appointment fell in between the those slots. Unfortunately, when the time display is from say 23:00 hrs to 01:30 hrs the top slot number is 46 and the bottom one is 3, the check to see if the appointment has any part showing in the display is a bit more complicated than a simple in between test.

This bug was fixed and tested in less than fifteen minutes.

4.4.2 Testing

The modularity of the execution modules proved to be very useful in testing. Except for the agenda execution module which needed interaction with other modules in order to be tested, most modules could be tested independently. For example, the date/time execution module contains the two most complicated activities, Julday and Caldat (Julday transform a Gregorian calendar date into the Julian day number and Caldat reverses the process - Caldat was only used for debugging purposes). To test these activities it was easy to write a small program that allow a date to be entered, return the Julian date, add a value to it, and get the new Gregorian date back. The same program was also used to test on which day the first of a given month and year fell (i.e. Monday, Tuesday, etc.).

In other words the methodology provides for modular testing of the system. The fact that the methodology leads to explicit coordination through its managers makes tracing of the program execution very easy. This is in contrast with other multitasking or object based programming methods where the code for coordination and execution are intermixed. In such programming methods the coordination is implicit and it is very difficult to relate coordination to the system design.

The steps of the outlined design procedure also provide an automatic documentation of the system and of the design decisions that were made. A tool which proved to be very useful was the system module file 'system.m', which contains a

complete description of the system; a list of procedures and functions in each module. This file is easy to make as one goes along the design process, and it provides a consistent documentation of the system. This file was also very helpful when modifying the system; it was very easy to establish which modules had to be changed.

This example, as expected, indicated that the Restricted Object Based methodology results in systems that are implemented from relatively small modules. As it can be seen in appendix A, all the managers and most execution modules' code, including comments, is one to four pages long. The two largest modules are; the agenda execution module and the window execution module are 12 and 23 pages respectively. Furthermore, these modules satisfied the characteristics of cost efficient design, laid out by Yourdon [31] which stated that the system modules should be: "easily related to the application", "manageably small" and "solvable and correctable separately" (also discussed in chapter III section 3.3 Characteristic of Resulting System).

CHAPTER V

EXAMPLE OF SYSTEM UPDATE

5.1 Modification Overview

One of the great advantages of the Restricted Object Based design methodology is the simplicity by which a system can be updated or modified. The separation of concerns between coordination and execution provided by this methodology enables updates to affect only a restricted portion of the original system. In this chapter we will describe the procedures to follow when doing updates by discussing three different examples:

- a. adding passwords, privacy and message recording;
- b. adding the time of day display (clock); and,
- c. providing a two month planning option.

5.2 Password, Privacy and Messages

Because the personal agenda can be started by any individual having access to the system it resides on, it was decided to modify the agenda to allow for privacy and messages. This modification provided limited information access based on three levels; a user level, a secretary level and a general public level.

5.2.1 Additional Requirements

The additional requirements for the updates requested in 5.2 above are as follow:

The personal agenda system provides three different access levels:

- a. Level 1 - user level access via a user password.
- b. Level 2 - secretary level access via a secretary password.
- c. Level 3 - general public level accessible with no passwords.

At level 1, all operations of the system are available. That is the user can add, delete, or modify appointments, and all appointments and their comments are displayed. When adding an appointment the user has the option of marking this appointment 'Private' by terminating the Add Function with 'AltP' or pressing the <PRIVATE> button. Marking an appointment private means that only the level 1 can see its comment.

At level 2, only the comments of appointments not marked private are shown. Private appointments are displayed as reserve time only by being highlighted. Only two functions are available at this level: Message and Quit. The Message Function opens a window in the lower left corner of the screen where the secretary may record a short message to be read by the user when he restarts the system.

At level 3, only the time reserved for appointments are displayed, no appointment comments are shown. This level has the same functions as level 2; Message and Quit.

The message buffer can record three lines of thirty-two characters each. These lines are displayed in the message window, which has a cyan background and a double line frame. The window is opened for editing only on a <MESSAGE> event. The editing key set is the same as for text box editing (Table II of requirements specification in Section IV) with the exception of the changes shown in Table IV.

Key	Action
'Enter'	Go to next line
Uarr	Move cursor one line up
Darr	Move cursor one line down
CtrlE	End message editing

Table IV Changes to editing key set for message editing.

At system start up, if a message is present the message window and the message buffer contents are displayed but the message edit mode is not activated unless the message event is initiated.

5.2.2 Update Procedure

The easiest way of looking at this update is to separate it into three smaller updates: 1 - add passwords, 2 - add privacy, and 3 - add message recording.

5.2.2.1 Adding Passwords

Adding passwords to the system means:

1. Being able to save and retrieve the passwords.
2. Querying the user at system start up for access password.
3. Providing the level 1 user with a change password function.
4. Restricting system information and function access according to start up level access.

The first step in doing an update is to list the expected changes to the system.

These are as follows:

- a. A new manager is required to respond to the PASSWORD event, allowing users to change the passwords.
- b. A new window is required for password entry or password editing
This requires the inclusion of a new service to the "window execution module".
- c. The "initialize manager" needs to be changed to request the password at system start-up.

NOTE: for upward compatibility, it should also provide initial password set up.

- d. The "agenda execution module", the "save_agenda" and "retrieve_agenda" activities should be changed to encrypt or decipher the passwords in the 'AG##.\$\$\$' file.
- e. The "system manager" should be changed to enable or disable functions according to the access level.
- f. The "fill_comment" activity in the "window execution module" should be modified to allow or disallow displaying of appointment comments according to access level
- g. Add the <PASSWORD> button to the command menu, this includes adding the coordinate to the "window execution module" "coordinate_to_event" activity and the 'P' key to the "keyboard execution module" "key_to_event" activity.
- h. Add two new global variables Ux[10] and Sx[10] to retrieve the user and secretary passwords.

From the above it is easy to see that one needs to modify three execution modules and two managers, and write one new response manager. The execution modules, modified services and activities are listed in Table V below.

Execution Module	SERVICE	Activity
window	OPEN_PASS*	open_password_window*
	OPEN_NEW_PASS*	
	OPEN_NOPASS*	
	WRITELN*	message_print*
	COORD_TO_EVENT	coordinate_to_event
agenda	FILLDAY	fill comment
	SAVE	save_agenda
	RETRIEVE	retrieve agenda
keyboard	KEY_TO_EVENT	key_to_event

(* indicates a new service or a new activity)

Table V Modified execution modules.

The managers that need changes are the "initialize manager" and the "system manager". In the "initialize manager" one has to add a password request sub-process right after retrieving the agenda data, instead of doing a simple "set_appointments" and "set_variables". The password request sub-process in Process Activity Language is as follow:

```

...
Case
((retrieve agenda) and (Ux is not empty)):
  open_password_window(OPEN_PASS);
  edit_string(password);
Case
  (password = Ux):
    level := 1;
  (password = Sx):
    level := 2;
  (default):

```

```

        level := 3;
    End Case;
    close_window;
    (agenda not retrieved or Ux empty):
    open_password_window(OPEN_NOPASS);
    edit_string(Ux);
    edit_string(Sx);
    level := 1;
    close_window;
End Case;
...

```

The changes to the "system manager" required simply to add a new table lookup for the PASSWORD event and level access verification to the 'add', 'delete' and 'modify' commands table lookup. The changes expressed in Process Activity Language are as follow:

```

...
Case
(ch = ADD):
    Case
        (level = 1):
            add_manager;
    End Case;
(ch = DELETE):
    Case
        (level = 1):
            delete_manager;
    End Case;
(ch = MODIFY):
    Case
        (level = 1):
            modify_manager;
    End Case;
...
(ch = PASSWORD):
    Case

```

```

        (level = 1):
            password_manager;
    End Case;
...

```

For the new response manager, one simply follows the procedures laid out in sections 3.2.1.1 and 3.2.3.1 of chapter III. The Process Activity Language definition of the new manager is as follow:

```

Process new_password_manager;
Sequence
    open_password_window(OPEN_NEW_PASS);
    edit_string(new_user_password);
Case
    (new_user_password = Ux):
        message_print; /* print second message */
        edit_string(Ux);
        message_print; /* print third message */
        edit_string(new_secretary_password);
        Case
            (new_secretary_password != Sx):
                sx := new_secretary_password;
        End Case;
    End Case;
close_window;
set_appointments;
fill_day_window;
hide_cursor;
End Sequence;
END.

```

Testing of this change was simple, it was done by asking the following questions:

- a. Is the change upward compatible (i.e. can I retrieve data which was saved without passwords)?
- b. Can one enter each of the three access level and display the information which is allowed at that level?
- c. Can one save and retrieve the passwords?
- d. Can one change the passwords?
- e. Are the proper command functions accessible at the different access levels?

This change was completed from requirements specification to total testing in 4 hours.

5.2.2.2 Adding Privacy

Adding privacy means:

- 1. Being able to mark an appointment as private.
- 2. Being able to save and retrieve this fact.
- 3. Restricting a private appointment's comment to be viewed at level 1 only.

The following is the list of the expected changes to the system:

- a. Add a private variable to the "appointment record".
- b. Include in the "add event manager" the possibility of terminating with a `<PRIVATE>` event.
- c. Add to the "insert activity" of the "agenda execution module" the capability of inserting the `<PRIVATE>` variable value. Include this change to parameter list of the insert activity.
- d. Add the PRIVATE event to the 'event.h' file.
- e. Add privacy to the "save_agenda" and "retrieve_agenda" activities of "agenda execution module". Remember upward compatibility must be kept (i.e. retrieving agenda data saved without private value must be possible).
- f. The "fill_comment" activity of the "window execution module" must now check for privacy as well as access level before displaying appointment comments.
- g. Add the `<PRIVATE>` button to the add function window. This includes adding the coordinate to the "coordinate_to_event" of the "window execution module" activity and the 'AltP' key to the "key_to_event" activity of the "keyboard execution module".
- h. Ensure that the "modify event manager" does not modify the appointment privacy value.

From the above list one can see that there is a need to modify three execution modules and two event managers. The execution modules, services and activities modified are listed in Table VI.

Execution Module	SERVICE	Activity
agenda	SAVE	save_agenda
	RETRIEVE	retrieve_agenda
	INSERT	insert_appointment
Window	OPEN_ADD	open_function_window
	COORD_TO_EVENT	coordinate_to_event
	FILLDAY	fill_comment
Keyboard	KEY_TO_EVENT	key_to_event

Table VI Modified execution modules for privacy update.

The modifications to the two event managers consist of adding a 'case' statement in the 'wait for event' section of the "add_appointment manager" (a) and to save the appointment's private variable once the modify section is entered (after the first OK) in the "modify_appointment manager" (b). The modifications expressed in Process Activity Language are as follow:

```

a. ..
    (event = OK or event = PRIVATE);
    Case
        (event = PRIVATE).
            mark_appointment_private;
    End Case;
...
Until (event = OK) or (event = CANCEL) or
    (event = PRIVATE);
...

b. ...
    save_appointment_private_value;
    ...
    set_modified_appointment_private_value;
    ..

```

These changes were very simple to test. One only had to enter an appointment and mark it private, then go to the level 2 and level 3 access and verify that its comment is not displayed. The time needed to implement and test this change was only one hour.

5.2.2.3 Adding Message Recording

Adding message recording and display means:

1. Being able to open a new window.
2. Being able to edit a three line message buffer.
3. Being able to save and retrieve this message.

This may sound simple to do, but it involves a few major changes. The list of expected changes is as follow:

- a. Define a new message manager to allow for editing of the message.
- b. A new activity `open_message_window` is expected to be required. This includes a new service to the "window execution module".
- c. Add a new global variable of type character (i.e. `char mess_text[3][33]`).
- d. The "save_agenda" and "retrieve_agenda" activities in the "agenda execution module" must be able to handle the message. Upward compatibility must be kept. Also messages may be empty.
- e. Add the <MESSAGE> button to the day window. This includes adding the coordinate for this button to the "window execution module" "coordinate_to_event" activity and adding the 'S' key to the "keyboard execution module" "key_to_event" activity. NOTE: 'S' was chosen because 'M' is already taken by modify function.
- f. Add the event MESSAGE to the 'event.h' file and to the "system manager" lookup table.
- g. Make a new `edit_message` activity in the "string execution module". This new activity is especially aimed at editing the new three line by thirty-two characters 'mess_text' buffer.

From the above list one can see that four execution modules as well as the system manager need to be modified and one new event manager was required. The execution modules' services and activities modified are listed in Table VII.

Execution Module	SERVICE	Activity
agenda	SAVE	save_agenda
	RETRIEVE	retrieve_agenda
Window	OPEN_MESS*	open_message_window*
	INITIALIZE	init_agenda_windows
	COORD TO EVENT	coordinate_to_event
String	EDIT_MESS*	edit_message*
Keyboard	KEY_TO_EVENT	key_to_event

(* indicates a new service or a new activity)

Table VII Modified execution modules for message update.

The change in the "system manager" consisted in adding the new MESSAGE event table lookup. In Process Activity Language this change corresponds to adding the following two lines:

```
...
(ch = MESSAGE):
    message_manager;
...
```

The new message manager in Process Activity Language is as follow:

```
Process message_manager;
Sequence
    open_message_window;
    edit_message;
    close_window;
    set_appointments;
    fill_day_window;
    hide_cursor;
```

End Sequence;
END.

Design, implementation and testing of the message recording update took one programmer six hours to complete. Three of these hours were spent in testing and debugging alone, as there were many combinations that had to be tested.

5.3 Time of Day Display

An update to the requirements specification for adding a running clock to the day window is as follow:

- a a running clock providing the time of day in the 24 hours format (i.e. 15:32) is displayed in the top right corner of the day window (beside the date header). This clock should be updated every minute - the system's time is used as the time of day.

5.3.1 Update Procedure

This update is a lot simpler to implement than the previous one. There exists already an activity providing us with the system's time ("system_time" in "date/time execution module"). The only question was where should the clock update take place? The obvious answer is in the "system manager", since it is the overall system coordinator. Therefore the list of expected changes is limited to the "system

manager", and the creation of a new clock manager. No new activity is needed since the clock can be updated using low level processing. The new manager is as follow:

```

Process update_clock;
Sequence
  temp := system_time;
Case
(clock != temp):
  clock := temp;
  cputs(clock);
  hide_cursor;
End Case;
End Sequence;
END.

```

The changes, in Process Activity Language to the "system manager" was limited to adding a clock update call inside the the two event waiting loops. The new event waiting loops in Process Activity Language now look as follow:

```

do {
  update_clock();
  ch = user_interface(EVENT);
} while ( ch == NOEVENT );

```

This update was incorporated and tested in less than thirty minutes.

5.4 Planning Option

The planning option allows the user to change the mode of operation and view the days with appointments over two consecutive months. To return to the normal

operating mode, the user selects any day from the two months using the mouse or the key commands.

This update requires major changes to the original personal agenda system. So far, the system has used a single background template with an overlay window approach for the various operations. The two month planning update requires the system to switch its background from the month and day window template to a two month window template and back.

5.4.1 Additional Requirements

The change in display from the two main windows (month and day) to two month windows is done by pressing the '<PLAN>' button or the 'L' key on the keyboard. In the 'planning' mode, the two months displayed are consecutive in time with the first of them displayed in the left window. Every day, within the two displayed months, with at least one appointment is highlighted (i.e. lightgray background with white lettering).

The current day, the day which will be displayed on return from this mode to the 'normal' mode, is flashing. The following operation can be performed in the 'planning' mode:

- a. Move to the next day (i.e. current day plus one) in the display by using the right arrow (Rarr) key. Moving to the next day from the last day of one month brings you to the first day of the other displayed month (move in a circular fashion).
- b. Move to the previous day (i.e. current day minus one) in the display by using the left arrow (Larr) key. Moving from the first day of one month brings you to the last day of the other displayed month (move in a circular fashion).
- c. Change the displayed months up by one (i.e. next month) by using the page up (PgUp) key or by pressing the '<NEXT>' button at the bottom of the left month window. This function makes each month window go up one month.
- d. Change the displayed months down by one (i.e. previous month) by using the page down (PgDw) key or by pressing the '<PREV>' button at the bottom of the left month window. This function makes each month window go down one month.

- e. Return to normal mode by pressing the 'Enter' key to return to the flashing day schedule or by selecting a day with the mouse, in any of the two months, to return to that day.

5.4.2 Expected Changes

This particular modification can be done in two different ways. The first approach is to add a parameter to the various activities to specify the mode of operation in which the system is currently operating. This requires that all managers using the affected activities be modified accordingly. The second approach is to use a global variable 'mode' which would indicate to the activities the current system operating mode. This way only those managers changing the mode of operation need to be modified. This second approach was used for this update, and the list of expected changes for it is as follow:

- a. A new global variable 'mode' is required to indicate if the system is in 'normal' or 'planning' mode of operation. Add 'NORMAL' and 'PLANNING' definition to 'event.h' file.
- b. A new two month manager is needed to initialize the 'planning' mode display.

- c. A new `init_two_month_windows` activity in the "window execution module" is required to initialize the two month window template.
- d. The "`fill_month`", "`fill_days`", "`set_appointment`" and "`highlight_day`" activities need to be modified to allow for the two different modes of operations.
- e. A new `make_day_flash` activity in the "window execution module" is needed to make the current selected day flash or stop the previous current selected day from flashing.³
- f. The "month up manager" and the "month down manager" need to be modified to include the 'planning' mode operation.
- g. The "next day manager" and the "previous day manager" need to be modified to allow movement of current day through both months.
- h. Modify the "system manager" to allow execution of the new PLAN event. Also add this event to the 'event.h' file.

³ This step became apparent only after the first iteration of the entire update process.

- j. Add an '<PLAN>' mouse button to the day window in normal mode.
This change includes adding the coordinate of this button to the "coordinate_to_event" activity of the "window execution module" .
Also add the 'L' key to the "key_to_event" activity of the "keyboard execution module" .
- k. Modify the "select_day" activity to allow selection from both month windows in 'planning' mode. This activity should also return the mode to 'normal' and re-initialize the month and day window to the selected day.
- l. Add the 'Enter' key to the "key_to_event" activity of the "keyboard execution module" for the 'planning' mode key set. This key initiates a "SELECT_DAY" event on the current flashing day.
- m. The following global variables are needed:
 - i. Integer select; indicate which month window is selected.
 - ii. Integer into; indicate in which month window is the current day flashing.

iii. Integer `next_month_offset`; indicate the offset of the first day of the month displayed in the second (right) month window.

iv. Integer `next_month_size`; indicate the number of days in the month displayed in the second (right) window.

n. The "clock manager" should be changed so that the clock is not displayed when in 'planning' mode.

From the above list it is easy to see that one needs to modify two execution modules and seven managers, and write one new event manager. The execution modules, services and activities modified are listed in Table VIII.

Using the design procedures defined in section 3.2.1.1 and 3.2.3.1 one can define the new 'two month manager'. The Process Activity Language for this process, including the starting event definition is as follows:

PLAN: Initialize two month window display for planning mode. This event is initiated by the 'L' key or by the <PLAN> mouse button in the day window of NORMAL mode.

```
Process two_month;
Sequence
  init_two_month_windows;
  mode := PLANNING;
```

```

select := 1;
into := 1;
im := current_month + 1;
Case
  (im = 13):
    im := 1;
    iy := current_year + 1;
  (default):
    iy := current_year;
End Case;
next_month_offset := next_month_first_day;
fill_month_window;
make_day_flash;          /* turn on flashing day */
set_appointments;
hide_cursor;
End Sequence;
END.

```

Execution Module	SERVICE	Activity
Window	TWO_MONTH*	init_two_month_windows*
	FILLMONTH	fill_month_window
	FILLDAY	fill_day_windows
	SETAPP	set_appointments highlight_day
	FLASH*	make_day_flash*
	COORD TO EVENT	coordinate to event
Keyboard	KEY_TO_EVENT	key_to_event

(* indicates a new service or a new activity)

Table VIII Modified execution modules for planning update.

Because the changes to the six event managers are too complicated to describe partially, the complete Process Activity Language specification of each of them is presented bellow.

```

a. Process next_day;
  Sequence
    Case
      (mode == NORMAL):
        Case
          (current_day < month_size):
            current_day + 1;
            set_current_variables;
            fill_day_window;
            hide_cursor;
          End Case;
        (mode == PLANNING):
          make_day_flash; /* turn off flashing day */
          Case
            (current_day into first month):
              Case
                (current_day < month_size):
                  current_day + 1;
                (current_day == month_size):
                  current_day := 1;
                  into := 2;
              End Case;
            (current_day into second month):
              Case
                (current_day < next_month_size):
                  current_day + 1;
                (current_day == next_month_size):
                  current_day := 1;
                  into := 1;
              End Case;
            End Case;
          flash_day := current_day;
          make_day_flash; /* turn on flashing day */
          hide_cursor;
        End Case;
      End Sequence;
    END.

```

b. Process previous_day:

Sequence

Case

(mode = NORMAL):

Case

(current_day > 1):

current_day - 1;

set_current_variables;

fill_day_window;

hide_cursor;

End Case

(mode = PLANNING):

make_day_flash; /* turn off flashing day */

Case

(current_day into first month):

Case

(current_day > 1):

current_day - 1;

(current_day = 1):

current_day := next_month_size;

into := 2;

End Case;

(current_day into second month):

Case

(current_day > 1):

current_day - 1;

(current_day = 1):

current_day := month_size;

into := 1;

End Case;

End Case;

flash_day := current_day;

make_day_flash; /* turn on flashing day */

hide_cursor;

End Case;

End Sequence;

END.

c. Process month_up;

Sequence

Case

(current_month = 12):

current_month := 1;

current_year + 1;

```

    (current_month < 12):
        current_month + 1;
End Case;
Case
    (mode == PLANNING):
        next_month_offset := next_month_first_day;
End Case;
month_offset := month_first_day;
fill_month_window;
current_day := 1;
into := 1;
Case
    (mode == NORMAL):
        set_appointments;
        set_current_variables;
        fill_day_window;
    (mode == PLANNING):
        set_appointments;
End Case;
flash_day := 1;
make_day_flash; /* turn on flashing day */
hide_cursor;
End Sequence;
END.

```

d. Process month_down;

Sequence

```

Case
    (current_month == 1):
        current_month := 12;
        current_year - 1;
    (current_month < 12):
        current_month - 1;
End Case;
Case
    (mode == PLANNING):
        next_month_offset := month_offset;
End Case;
month_offset := month_first_day;
fill_month_window;
current_day := 1;
into := 1;
Case
    (mode == NORMAL):

```

```

        set_appointments;
        set_current_variables;
        fill_day_window;
    End Case;
    flash_day := 1;
    make_day_flash; /* turn on flashing day */
    set_appointments;
    hide_cursor;
End Sequence;
END.

```

```

e. Process select_day;
   Sequence
     Case
       (mode = NORMAL):
         Case
           (calendar_square in month):
             current_day := selected_day;
             set_current_variables;
             fill_day_window;
             hide_cursor;
         End Case
       (mode = PLANNING):
         init_agenda_windows;
         mode := NORMAL;
         Case
           (select day into second month):
             month_offset := next_month_offset;
             month_size := next_month_size;
             current_month + 1;
         Case
           (current_month = 13):
             current_month := 1;
             current_year + 1;
         End Case;
       End Case;
     fill_month_window;
     reset_myclock;
     Case
       (calendar_square in month):
         set_appointments;
         set_variables;
         fill_day_window;
         hide_cursor;
     End Case;
   End Sequence;
End Process;

```

```

        End Case;
    End Case;
End Sequence;
END.

```

```

f. Process update_clock;
    Sequence
    Case
        ( mode == NORMAL )
            temp := system_time;
            Case
                (clock != temp):
                    clock := temp;
                    cputs(clock);
                    hide_cursor;
            End Case;
        End Case;
    End Sequence;
END.

```

The Changes to the system manager are limited to adding the new event table lookup and limiting operation access while in 'planning' mode; this is done the same way as for the password update. The Process Activity Language of the changes for this manager are as follows:

```

...
(ch = ADD):
    Case
        ((mode != PLANNING) and (level == 1)):
            add_manager;
        End Case;
(ch = DELETE):
    Case
        ((mode != PLANNING) and (level == 1)):
            delete_manager;
        End Case;
(ch = MODIFY):
    Case

```



```

      ((mode != PLANNING) and (level = 1)):
        modify_manager;
    End Case;

```

...

```

(ch = SCROLL_UP):
  Case
    (mode != PLANNING):
      scroll_up_manager;
  End Case;
(ch = SCROLL_DOWN):
  Case
    (mode != PLANNING):
      scroll_down_manager;
  End Case;

```

...

```

(ch = PASSWORD):
  Case
    (mode != PLANNING):
      new_password_manager;
  End Case;
(ch = MESSAGE):
  Case
    (mode != PLANNING):
      message_manager;
  End Case;
(ch = PLAN):
  Case
    (mode != PLANNING):
      two_month_manager;
  End Case;
(ch = QUIT):
  Case
    (mode != PLANNING):
      quit_manager;
  End Case;

```

...

The complete modification, implementation and testing of the two month planning option required a single programmer eleven hours to complete.

5.5 Comments

As indicated by the above examples, modifications are easy to implement. One starts by making an expected change list which follows directly from the updated requirements. This list provides the programmer with a step by step plan for implementation. Most changes affect activities, this corresponds to adding or modifying services in existing execution modules. In some systems there may arise the need to define new execution modules. The changes in the managers (adding or modifying sub-processes) generally involved a few lines. In most cases one needs to add a new response manager(s) which can be written in the same way as outlined in sections 3.2.1.1 and 3.2.3.1.

In the same way as it was mentioned in section 4.4.2 these changes were easy to make because of the separation of coordination and execution. Furthermore, since the modules of the system are closely related to the application it is easy to pin point the area where changes should occur. To implement and test all of the changes outlined in this chapter required slightly less than three man days.

To test modifications one only requires differential testing of the system as only the new services need to be checked; as a rule the previously existing services

need not be checked out. However, one of the bugs which appeared as a result of updates was surprising. After adding the password, the mouse operation became very erratic. It turned out that in modifying the "initialize manager" for the password entry, the password window was not closed after reading the password. It was wrongly assumed that since the screen was cleared after entering the password, the password window became inexistent and that there was no need for the CLOSE window service. The "coordinate_to_event" activity keeps track of the windows opened and closed to translate the mouse coordinates into the proper event. Not closing the window meant that this activity was assuming a window was still open, when in fact there were none. This bug was fairly simple to trace, since it was known that the system worked before and that the only thing that might affect the mouse would be the "coordinate_to_event" activity of the "window execution module". A simple trace of this activity provided the solution.

As with the design procedures, following the modification steps provides self documentation of the revised system. Keeping previous version documentation can provide an easy to follow derivation of changes from one version to the next.

CHAPTER VI

CONCLUSION

6.1 Thesis Summary

The research done in this thesis has addressed the problem of designing practical cost effective general purpose interactive software, also referred to as shrink-wrap software. In chapter I it was shown that shrink-wrap software is different enough to warrant the development of a design methodology of its own. In chapter II we considered the software development process, categorized the present day graphical design specifications and saw that:

- a. The data flow graphs are better suited for systems with large amount of data manipulation and little concurrencies.
- b. The control flow graphs are better suited for systems with concurrent ordered sequences of operations.
- c. The relation diagrams are better suited for systems where resources manipulation and allocation is predominant.

From the characteristics of shrink-wrap software we deduced that the proper design methodology should be a combination of the general control flow approach and the object based approach of relation diagrams.

The event driven nature of shrink-wrap software made a design methodology based on the multiactivity paradigm a sensible choice for its design. In chapter III we saw that in shrink-wrap software we can separate the coordination from the activity execution even if both are running on the same processor. This same chapter outlined the restricted object based design methodology ; a design method which takes advantage of the reusability of object based programming techniques and the behavioral aspect of the Process Activity Language. The restricted object based design methodology is implemented in three simple stages:

- a. In the preliminary stage one must translate the requirements into a set of event responses, constraints and data structure.
- b. In the next stage, the grouping of activities is done according to user defined attributes, resulting in the definition of the various execution modules of the system.
- c. In the last stage, the various managers are defined. The system manager and the response managers are derived directly from the event

responses of the preliminary stage and the event manager is derived from the interface requirements.

These three steps are iteratively repeated until the final resulting system's design is obtained. Once the complete design specification is obtained, all that is required is to implement small modules. Each module can then be tested separately before the complete system is validated. The restricted object based design methodology is a complete methodology from requirements specification to testing and updating.

In chapter IV, it was demonstrated that the restricted object based design methodology works by implementing a personal agenda system based on this scheme. Although it is very hard to get exact figures on the time required to develop the example system, as both the methodology and the example were done concurrently, it is estimated that for a design engineer familiar with this methodology and having good programming skills in 'C', the design of such a system would take:

- a. Stage 1 - formalizing the requirements - 3 to 4 man days.
- b. Stage 2 - definition of the execution modules, including the modifications which could arise from iteration of the design procedures - 3 to 4 man days.

- c. Stage 3 - the definition of the various managers - 1 to 2 man days.
- d. Actual implementation or coding of the activities, including testing and debugging - 10 to 12 man days.

That is, one could develop a basic system which is fully documented and easy to modify within 17 to 22 man days. Because of the separation of concerns (coordination and activity execution) and the small size of the resulting modules, it is also estimated that the design complexity will grow proportionally with the system complexity. That is, it is our belief that this methodology scales up well.

Chapter V showed that a system designed using this methodology was easy to modify and update, by providing three different updates to the personal agenda system. To modify a system one simply needs to translate the additional requirements into a list of expected changes before formalizing them into additional processes or sub-processes in Process Activity Language. All of the updates found in this chapter were implemented and tested in less than three man days. In terms of lines of code per hour, the system was implemented at a rate of about 20 lines per hour, while the updates were done at the rate of 40 lines per hour (see Table IX). From discussion with other software designer we believe that these numbers represent good productivity. The difference between implementation and updates is accountable, partly because the system implementation estimate is optimistic, but more so by the

	Original version 1.0		Updated version 2.1	
	# of lines	# of code	# of lines	# of code
System.m	143	67	143	67
Char.h	157	143	157	143
Event.h	38	22	44	28
System.mgr	130	89	160	117
Event.mgr	57	31	57	31
Add.mgr	108	87	116	93
Del.mgr	116	98	121	101
Mod.mgr	218	187	233	198
Monday.mgr	37	26	44	33
Monday.mgr	37	26	38	47
Monday.mgr	24	13	37	45
Friday.mgr	23	13	34	44
Saturday.mgr	26	14	38	46
Sunday.mgr	47	34	47	34
Sunday.mgr	47	34	47	34
Init.mgr	64	43	99	78
Quit.mgr	19	9	19	9
Newpass.mgr	0	0	37	27
Mass.mgr	0	0	22	14
Clock.mgr	0	0	28	18
Twom.mgr	0	0	39	27
Agenda.em	533	437	634	518
Error.em	89	63	89	63
Keyboard.em	128	80	169	118
Mouse.em	203	131	203	131
String.em	226	149	332	233
Time.em	221	144	223	190
Window.em	900	729	1228	1027
Window.c	218	157	218	157
Agenda.dat	278	191	293	202
P.A.c	6	5	6	5
TOTAL	4111	3032	5033	3866

Table IX. Size of files of the personal agenda system.

fact that the documentation is easy to update and the sections to be updated are easily located, easy to understand and small (i.e. manageable).

From these later two chapters we believe that we have shown the validity of the restricted object based design methodology. The design resulted in a highly modular system where the modules are easily related to the application, are manageably small and are solvable, correctable and modifiable separately. Furthermore, the methodology provides for self documentation which is easy to understand and provides a means of tracing how the various decisions were made and how the various versions were derived.

6.2 Future Work

Now that the validity of this design method has been demonstrated, we believe it is time to develop some tools for it. First we consider the possibility of creating generic execution modules or activity libraries like the one used for the window execution module in chapter IV. Generic execution modules would be built for the development of a Graphical User Interface like the one created at [33], where the execution module would contain general purpose graphical activities. Further, generic execution modules could be created for the development of a robot simulator for robotic research, these execution modules would contain the various behaviour, movements or actions of a robot.

A second tool which is considered is the creation of a Process Activity Language editor like the one found in [29]. Instead of translating the processes for a simulator, this editor could detect syntax errors and be the front end of an automatic code generator for all the managers.

Apart from development of tools it would be interesting to find out how this methodology applies to more specialized interactive software that cannot be categorized as shrink-wrap software. Here one could consider systems which are centralized like a company inventory system or distributed like a banking machine system or an airline reservation system. As long as a system can be translated into events and event responses, the methodology would most likely lead to a suitable solution.

To conclude, at times during the development of the example, it was felt that there was an explosion of code in comparison to a conventional structured programming method, where coordination and execution are intermixed. This explosion of code may not necessarily exist or actually mean that the resulting executable code is larger. Because of the highly procedural nature of the resulting system, the executable code could be much smaller, similar to the result obtained when comparing segmented microprogramming (which is a multiactivity system) to the regular microprogramming [34]. To get meaningful data would require the

development of a number of systems by different programmers using different design methods.

REFERENCES

- [1] W.M. Gentleman, and M. Wein, "Mass Market Computers for Software Development", Processings of the 1991 ACM Symposium on Personal and Small Computers, Toronto, Ont., June 12-14, 1991.
- [2] W.M. Gentieman, "Engineering the Process of Software Evolution", presentation given at a NATO Conference, Royal Signal Radar Establishment, Malvern, England, 1990.
- [3] S. Feldman, M. Gentleman, "Portability - A No Longer Solved Problem", Computing Systems, Vol 3 No. 2 Spring 1990, pp. 359-380.
- [4] Mary Shaw, "Prospects for an Engineering Discipline of Software", IEEE Software, November 1990.
- [5] Webster's New World Dictionary, Third College Edition Webster's New World, New York 1988.
- [6] M. Krieger, "The Multiactivity Paradigm: An Approach for the Design of Embedded Systems by Application Specialists", a paper submitted to IEEE Transactions on Systems, Man and Cybernetics.

- [7] G. Brookshear, "Computer Science an Overview", Third Edition, The Benjamin/Cummings Series in Computer Science, The Benjamin/Cummings Publishing Company Inc., Redwood City, California, 1991.
- [8] G. Polya, "How to Solve It: A new aspect of mathematical method", Princeton, N.J., Princeton University Press 2d ed. 1973.
- [9] R.S. Pressman, "Software Engineering A Practitioner's Approach", Second Edition, McGraw-Hill Series in Software Engineering and Technology, McGraw-Hill Book Company, 1987.
- [10] P. Zave, "The Operational versus The Conventional approach to Software Development", Communications of the ACM, Vol 27, No. 2, Feb 1984, pp. 104-118.
- [11] Kiwamoto, O. Shigo, "Unifying Dataflow and Control Flow based Modularization Techniques", Proceeding Fall Compcon Los Alamitos Cal. 1981, pp. 271-277.
- [12] B. Lent, "Data Flow Architecture for Machine Control", Research Studies Press Ltd, Somerset, England, John Wiley & Sons Inc., New York, 1989.
- [13] P.T. Ward, "The Transformation Schema: An Extension of the Data Flow Diagram

to Represent Control and Timing", IEEE Transactions on Software Engineering, Vol SE-12, No. 2, Feb 1986, pp. 198-210

- [14] Georg Reader, "A Survey of Current Graphical Programming Techniques", IEEE Computer, Vol 18, No. 8, August 1985.
- [15] J.L. Peterson, "Petri Net Theory and The Modeling of Systems", Prentice-Hall, Inc, Englewood Cliffs, N.J., 1981.
- [16] G. Michel, C. Lurgeau, B. Espiau, "Les Automates Programmables Industriels", Dunod, Paris, France, 1979.
- [17] R. Buhr, and R. Casselman, "Architectures with Pictures", OOPSLA'92, SIGPLAN Notices, Vol. 27, No. 10, ACM, Vancouver, Oct. 1992.
- [18] W.P. Stevens, G.J. Myers, and L.L. Constantine, "Structured Design", IBM Systems Journal, Vol 13, No. 2, May 1974, pp. 115-139.
- [19] E. Yourdon, "Managing the Structured Techniques", Second Edition, Prentice_hall, Inc., Englewood Cliffs, New Jersey, 1979.
- [20] G. Reader, "A survey of current Graphical Programming Techniques", Computer,

Vol.18, No.8, August 1985,
pp. 11-25.

- [21] A. Bowen, and R. Brown, "Systems Design: Volume II of VLSI Systems Design for Digital Signal Processing",
Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1985.
- [22] A. Buhr, "System Design with Ada", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1984.
- [23] G. Booch, "Object Oriented Design with Applications", The Benjamin/Cummings Publishing Company, Inc., Redwood City, California, 1991.
- [24] G. Booch, "Object-Oriented Development", IEEE Transactions on Software Engineering, Vol SE-12, No. 2, February 1986, pp. 211-221.
- [25] J.C. Kelly, "A Comparison of Four Design Methods for Real-Time Systems", 9th International Conference on Software Engineering, Monterey, California, March 1987.
- [26] D. Gelernter, and N. Carriero, "Coordination Languages and their Significance", Communications of the ACM, Vol. 35, No. 2, February 1992.

- [27] R. Joannis, "Specification and Design of Multiple Microprocessor Systems Using Process Activity Diagrams", MEng. Thesis, University of Ottawa, August 1986.

- [28] M. Krieger, "Multiactivity Paradigm for the Coordination of Flexible Manufacturing Systems (FMSs)", Computer Integrated Manufacturing System, Vol.6, No.3, August 93, pp. 195-203.

- [29] A. Chubukjian, "Extended PAL Editor An Example of an Interactive Real-Time System Specification Tool", MEng. Thesis, University of Ottawa, April 1990.

- [30] R.E. SanMartin, "The Multiactivity Paradigm in Rapid Prototyping of embedded system", M.A.Sc. Thesis, Ottawa-Carleton Institute for Electrical Engineering, 1990.

- [31] E. Yourdon and L. Constantine, "Structured Design Fundamentals of a Discipline of Computer Program and System Design", Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1979.

- [32] Dan Sharon, "Run-Time Kernel for Multiactivity Systems Using 'Process Activity Language (PAL)", MEng. Thesis, Carleton University, March 1991.

- [33] D. Jones, "CPAL: A Multiactivity Development System for Interactive Graphical User Interfaces", Engineering project at University of Ottawa, Dec 11, 1992.
- [34] S. Vaillancourt, "A Segmented Microprogrammed Processor", M.A.Sc. Thesis, University of Ottawa, 1980.
- [35] K.W. Christopher, Jr., B.A. Feigenbaum, S.O. Saliga, "Developing Applications using DOS", John Wiley and Sons, Inc. 1990.
- [36] K.S. Booth, W.M. Gentleman, "Symbiotic Computing: Opportunities and Complications", NRC report 33156
- [37] E.M. Prell, and A.P. Sheng, "Building Quality and Productivity into a Large Software System", IEEE Software Vol 1 Num. 3 July 1984, pp. 47-54.
- [38] P. Wegner, "Capital-intensive Software Technology Part 2: Programming in the Large", IEEE Software Vol 1 Num. 3 July 1984, pp. 24-32.
- [39] D. Harel et.al., "STATEMATE: A Working Environment for the Development of Complex Reactive Systems", IEEE Transactions on Software Engineering, Vol 16 No. 4, April 1990, pp. 403-413.

- [40] D. Hatley, I. Pirbhai, "Strategies for Real-Time System Specification", Dorset House Publishing, New York, 1987.
- [41] J. Baer, "Computer Systems Architecture", Computer Science Press, Rockville, Maryland, 1980.
- [42] S. Faulk, J. Brackett, P. Ward, and J. Kirby Jr., "The Core Method for Real-Time Requirements", IEEE Software, September 1992.
- [43] R.A. Harvey, "Process Activity Language Interactive Verification System", M.Eng. Thesis, University of Ottawa, May 1989.
- [44] Borland International, "Turbo C Reference Guides: Version 2.0", Borland International, Scotts Valley, California, 1988.
- [45] W. Press, B. Flannery, S. Teukolsky, and W. Vetterling, "Numerical Recipes the Art of Scientific Computing", Cambridge University Press, 1986.

APPENDIX A

Personal Agenda code listing

```

/* *****
/* file: SYSTEM.M */
/* *****

/* This file contains the list of procedures/functions of the */
/* entire agenda system. The name in comment is the file in */
/* which all procedures/functions, listed below it, appear. */
/* It is not necessary for the system to work, but it is a */
/* great reference document for debugging or modifying the */
/* program. */

/* system.mgr */
int system_manager(int service);

/* init.mgr */
void initialize_manager(void);

/* event.mgr */
int user_interface(int service);

/* month.mgr */
void month_up_manager(void);

/* month.mgr */
void month_down_manager(void);

/* select.mgr */
void select_day_manager(void);

/* nextday.mgr */
void next_day_manager(void);

/* preday.mgr */
void previous_day_manager(void);

/* scroll.mgr */
void scroll_up_manager(void);

/* scroll.mgr */
void scroll_down_manager(void);

/* clock.mgr */
void update_clock(void);

/* mess.mgr */
void message_manager(void);

/* newpass.mgr */
void new_password_manager(void);

/* twom.mgr */
void two_month_manager(void);

/* keyboard.em */
int keyboard_em(int service, char key);

/* mouse.em */
int mouse_installed(void);
void show_mouse_cursor(void);
void hide_mouse_cursor(void);
void move_mouse(int xcoord, int ycoord);
void get_mouse_action(int *but, int *hor, int *ver);
void set_mouse_cursor(int Ordchar);
int mouse_pressed(int button);
int mouse_released(int button);
int mouse_em(int service, char *parameter);

```

APPENDIX A
Personal Agenda code listing
138

```
long julday(int mm, int id, int yyyy);
void caldat(long julian, int *mm, int *id, int *yyyy);
void system_time(char *time_now);
void system_date(char *today);
void timer_value(char *val);
int mod(long x, int y);
int date_time_em(int service, char *parameters);

/* agenda.em */

PYears new_year(void);
PMonths new_month(void);
PDays new_day(void);
PAppt new_appointment(void);
void save_agenda(void);
void retrieve_agenda(void);
int insert_appointment(char *parameters);
int delete_appointment(char *parameters);
int agenda_em(int service, char *parameters);

/* string.em */

void setcursor(unsigned int shape);
void changecursor(int inmode);
void initcursor(void);
int editstring(char *s, int posx, int posy, int maxlength);
void string_em(int service, char *s, char *parameter);
int edit_message(void);

/* error.em */

void error_em(int error_no);

/* window.em */

void command_line(int x, int y);
void init_agenda_windows(void);
void open_function_window(int typ);
int coordinate_to_event(char *parameter);
void fill_month_window(void);
void fill_day_window(void);
int window_em(int service, char *parameter);
void open_message_window(void);
void message_print(char *param);
void open_password_window(int typ);
void init_two_month_windows(void);
void make_day_flash(int on);

/* windobj.c */

void save_screen(int x1, int y1, int x2, int y2, char *buffer);
void close_window(void);
void open_window(int x1, int y1, int x2, int y2, int bkgnd,
                 int frame, char *title);
void text_box(int x, int y, int length, char *title);
void highlight_box(int booram, int mode);
```

APPENDIX A Personal Agenda code listing 139

```

/*.....*/
/* file: CHAR.H */
/*.....*/

/***** Definition of input characters *****/

#define BackSp      8
#define Tab         9
#define Enter       13
#define Esc         27
#define STab        143 /* Shift Tab */
#define PgUp        201
#define PgDn        209
#define EndKey      207
#define Home        199
#define Ins         210
#define CtrlPgUp    138
#define CtrlPgDn    246
#define CtrlEnd     245
#define CtrlHome    247
#define Del         211
#define Larr        203 /* Left arrow */
#define Rarr        205 /* Right arrow */
#define Uarr        200 /* Up arrow */
#define Darr        208 /* Down arrow */
#define CtrlLarr    243 /* Control Left arrow */
#define CtrlRarr    244 /* Control Right arrow */
#define CtrlPrtc    242 /* Control PrintScreen */
#define CtrlA       1  /* Control A */
#define CtrlB       2  /* Control B */
#define CtrlC       3  /* Control C */
#define CtrlD       4  /* Control D */
#define CtrlE       5  /* Control E */
#define CtrlF       6  /* Control F */
#define CtrlG       7  /* Control G */
#define CtrlH       8  /* Control H */
#define CtrlI       9  /* Control I */
#define CtrlJ       10 /* Control J */
#define CtrlK       11 /* Control K */
#define CtrlL       12 /* Control L */
#define CtrlM       13 /* Control M */
#define CtrlN       14 /* Control N */
#define CtrlO       15 /* Control O */
#define CtrlP       16 /* Control P */
#define CtrlQ       17 /* Control Q */
#define CtrlR       18 /* Control R */
#define CtrlS       19 /* Control S */
#define CtrlT       20 /* Control T */
#define CtrlU       21 /* Control U */
#define CtrlV       22 /* Control V */
#define CtrlW       23 /* Control W */
#define CtrlX       24 /* Control X */
#define CtrlY       25 /* Control Y */
#define CtrlZ       26 /* Control Z */
#define AltA        158 /* Alternate A */
#define AltB        176 /* Alternate B */
#define AltC        174 /* Alternate C */
#define AltD        160 /* Alternate D */
#define AltE        146 /* Alternate E */
#define AltF        161 /* Alternate F */
#define AltG        162 /* Alternate G */
#define AltH        163 /* Alternate H */
#define AltI        151 /* Alternate I */
#define AltJ        164 /* Alternate J */
#define AltK        165 /* Alternate K */
#define AltL        166 /* Alternate L */
#define AltM        178 /* Alternate M */
#define AltN        177 /* Alternate N */
#define AltO        152 /* Alternate O */
#define AltP        153 /* Alternate P */
#define AltQ        144 /* Alternate Q */
#define AltR        147 /* Alternate R */
#define AltS        159 /* Alternate S */
#define AltT        148 /* Alternate T */
#define AltU        150 /* Alternate U */
#define AltV        175 /* Alternate V */
#define AltW        145 /* Alternate W */
#define AltX        173 /* Alternate X */
#define AltY        149 /* Alternate Y */
#define AltZ        172 /* Alternate Z */
#define Alt1        248 /* Alternate 1 */
#define Alt2        249 /* Alternate 2 */
#define Alt3        250 /* Alternate 3 */
#define Alt4        251 /* Alternate 4 */

```

APPENDIX A Personal Agenda code listing 140

```

#define Alt5      252 /* Alternate 5 */
#define Alt6      253 /* Alternate 6 */
#define Alt7      254 /* Alternate 7 */
#define Alt8      255 /* Alternate 8 */
#define Alt9      134 /* Alternate 9 */
#define Alt0      135 /* Alternate 0 */
#define Altminus  136 /* Alternate minus */
#define Altequals 137 /* Alternate equals */
#define F1        187
#define F2        188
#define F3        189
#define F4        190
#define F5        191
#define F6        192
#define F7        193
#define F8        194
#define F9        195
#define F10       196
#define F11       139
#define F12       140
#define sF1       212 /* Shift F1 */
#define sF2       213 /* Shift F2 */
#define sF3       214 /* Shift F3 */
#define sF4       215 /* Shift F4 */
#define sF5       216 /* Shift F5 */
#define sF6       217 /* Shift F6 */
#define sF7       218 /* Shift F7 */
#define sF8       219 /* Shift F8 */
#define sF9       220 /* Shift F9 */
#define sF10      221 /* Shift F10 */
#define sF11      141 /* Shift F11 */
#define sF12      142 /* Shift F12 */
#define CtrlF1    222 /* Control F1 */
#define CtrlF2    223 /* Control F2 */
#define CtrlF3    224 /* Control F3 */
#define CtrlF4    225 /* Control F4 */
#define CtrlF5    226 /* Control F5 */
#define CtrlF6    227 /* Control F6 */
#define CtrlF7    228 /* Control F7 */
#define CtrlF8    229 /* Control F8 */
#define CtrlF9    230 /* Control F9 */
#define CtrlF10   231 /* Control F10 */
#define CtrlF11   154 /* Control F11 */
#define CtrlF12   155 /* Control F12 */
#define AltF1     232 /* Alternate F1 */
#define AltF2     233 /* Alternate F2 */
#define AltF3     234 /* Alternate F3 */
#define AltF4     235 /* Alternate F4 */
#define AltF5     236 /* Alternate F5 */
#define AltF6     237 /* Alternate F6 */
#define AltF7     238 /* Alternate F7 */
#define AltF8     239 /* Alternate F8 */
#define AltF9     240 /* Alternate F9 */
#define AltF10    241 /* Alternate F10 */
#define AltF11    156 /* Alternate F11 */
#define AltF12    157 /* Alternate F12 */

/* mouse key values */

#define MUp       128 /* Mouse move up */
#define MDown     129 /* Mouse move down */
#define MLeft     130 /* Mouse move left */
#define MRight    131 /* Mouse move right */
#define MRightB   132 /* Mouse right button */
#define MLeftB    133 /* Mouse left button */

/* define the constant for mouse buttons recognition */

#define NoB       0 /* no button */
#define LeftB     1 /* Left button */
#define RightB    2 /* Right button */
#define BothB     3 /* Both buttons */

/* file EVENT.H */

/* This file contains the definition of the possible events for the
 * Agenda system.
 */
/*-----*/

/* mouse events */

```

APPENDIX A
Personal Agenda code listing
141

```
#define NOEVENT      -1
#define ADD          0
#define DELETE       1
#define MODIFY       2
#define MONTH_UP     3
#define MONTH_DOWN   4
#define SELECT_DAY   5
#define SCROLL_UP    6
#define SCROLL_DOWN  7
#define QUIT         8
#define START_TIME   9
#define END_TIME     10
#define COMMENT      11
#define CANCEL       12
#define OK           13
#define NEXT_DAY     14
#define PREVIOUS_DAY 15
#define ACTIVATE     16
#define NEXT_BOX     17
#define TIME_BOOKED  18
#define MESSAGE      19
#define PASSWORD     20
#define PRIVATE      21
#define PLAN         22

/*-----*/
/* Other useful definition */
/*-----*/

#define RESET 0
#define SET 1
#define NORMAL 3 /* normal operating mode */
#define PLANNING 4 /* planning operating mode */
```

[illegible]

143

[illegible]

144

[illegible]

```
typedef struct appt          /* appointment structure */
{
    char start time[6];
    char end time[6];
    char comment[36];
    int private;
    struct appt *next;      /* pointer to next appointment */
} *PAppt;

typedef struct days          /* day structure */
{
    char id[3];             /* date in characters */
    PAppt appointment;      /* appointment for that day */
    struct days *next;      /* pointer to next day */
} *PDays;

typedef struct months        /* month structure */
{
    char mm[3];             /* month number */
    PDays day set;         /* list of days with appointment that month */
    struct months *next;    /* pointer to next month */
} *PMonths;

typedef struct years         /* year structure */
{
    char iyyy[5];           /* year id in char */
    PMonths month set;      /* list of month with appointment that year */
    struct years *next;     /* pointer to next year */
} *PYears;
```

APPENDIX A
Personal Agenda code listing
145

```

/****** */
/* Agenda global variables */
/*-----*/

/* Agenda is the main structure */
PYears Agenda, cyear; /* cyear == current year pointer */
PMonths cmonth; /* current month pointer */
PDays cday; /* current day pointer */
PAppt cappointment; /* current appointment pointer */

/* current displayed data may not be part of the saved data */
/* i.e. in the case where a month has no entry. Therefore */
/* string are used to remember what is the displayed data. */

char current_year[5];
char current_month[3];
char current_day[3];
char current_time[6];
char display_time[6]; /* start of appointment window displayed time */
char ux[10]; /* user password */
char sx[10]; /* secretary password */
char mess_text[3][34]; /* text for message window */
char myclock[12] = ""; /* clock time display */

int month_size; /* number of days in the current month */
int month_offset; /* offset for the first day of the month */
int next_month_offset; /* offset for first day of second month in */
/* PLANNING mode */
int next_month_size; /* number of days in the second month in */
/* PLANNING mode */
int Alarm; /* appointment alarm boolean */
int flash_day; /* selected day flashing in the month window */
int level; /* access level permitted by the password */
int mode; /* agenda operating mode (NORMAL or PLANNING) */
int select; /* window selection in PLANNING mode */
int into; /* remember which month window has the current day */

FILE *ag; /* file pointer for agenda structure */

```

APPENDIX A
Personal Agenda code listing
146

```

/******
/* file: SYSTEM.MGR */
/******

/* The system manager is the main program. Note that it does not
/* provide services in itself (even if it has a service parameter).
/* The response managers request services from other response
/* managers through the system manager, this is why the parameter.
/* The main program request only that the system be started.

/******
/* service requested through the manager */
/*-----*/

#define EDITING 0
#define WTE_inst 1
#define MOUSED 2
#define SYSTEM 3

/******
/* include all system files */
/*-----*/

#include "agenda.dat"
#include "system.m"
#include "char.h"
#include "event.h"
#include "window.em"
#include "error.em"
#include "agenda.em"
#include "string.em"
#include "time.em"
#include "keyboar.em"
#include "mouse.em"
#include "inat.mgr"
#include "event.mgr"
#include "scrden.mgr"
#include "scrup.mgr"
#include "selday.mgr"
#include "monap.mgr"
#include "monden.mgr"
#include "preday.mgr"
#include "nextday.mgr"
#include "add.mgr"
#include "del.mgr"
#include "mod.mgr"
#include "quit.mgr"
#include "mess.mgr"
#include "newpass.mgr"
#include "clock.mgr"
#include "twom.mgr"

/******
int system_manager ( int service )
{
    int ch, terminate = 0;

    switch ( service ) {
        case EDITING: do
            ch = user_interface(TEXT);
            while ( ch == 0 );
            return(ch);
        case WTE_inst: do {
            update_clock();
            ch = user_interface(EVENT);
        } while ( ch == NOEVENT );
            return(ch);
        case MOUSED: return( user_interface(INITIAL) );
        case SYSTEM: initialize_manager();
            if( strlen(mess_text[0]) != 0 )
            {
                window_em(OPEN_MESS,"");
                window_number--; /* do not count as a window */
            }
            if( !user_interface(INITIAL) ) error_em(NO_MOUSE);
            do {
                do {
                    update_clock();
                    ch = user_interface(EVENT);
                } while( ch == NOEVENT );
            }
    }
}

```

```

switch ( ch ) {

    case ADD:    if( mode == PLANNING )
                  break;
                  if( level == 1 )
                      add_manager();
                  break;

    case DELETE: if( mode == PLANNING )
                  break;
                  if( level == 1 )
                      delete_manager();
                  break;

    case MODIFY: if( mode == PLANNING )
                  break;
                  if( level == 1 )
                      modify_manager();
                  break;

    case MONTH_UP: month_up_manager();
                  break;

    case MONTH_DOWN: month_down_manager();
                  break;

    case SELECT_DAY: select_day_manager();
                  break;

    case SCROLL_UP: if( mode == PLANNING )
                  break;
                  scroll_up_manager();
                  break;

    case SCROLL_DOWN: if( mode == PLANNING )
                  break;
                  scroll_down_manager();
                  break;

    case NEXT_DAY: next_day_manager();
                  break;

    case PREVIOUS_DAY: previous_day_manager();
                  break;

    case PASSWORD: if( mode == PLANNING )
                  break;
                  if( level == 1 )
                      new_password_manager();
                  break;

    case MESSAGE: if( mode == PLANNING )
                  break;
                  message_manager();
                  break;

    case PLAN: if( mode == PLANNING )
                  break;
                  two_month_manager();
                  break;

    case QUIT: if( mode == PLANNING )
                  break;
                  quit_manager();
                  terminate = 1;
                  break;
} /* end switch ( ch ) */
} while( !terminate );
} /* end switch ( service ) */
user_interface( SHUTDOWN ); /* remove the mouse cursor */
return(0); /* terminate program normally */
}

```

```

*****
/* file: EVENT.MGR */
*****

/* The event manager provides input service from the mouse and the
/* keyboard. The values of the characters or events return are in
/* integers. Control characters are define in file CHAR.H (which
/* is included in the execution modules). The system manager request
/* events to the event manager in the form of service. The possible
/* events are defined in the window execution module.

/*****
/* SERVICES PROVIDED BY USER INTERFACE
*****/

#define TEXT      0
#define EVENT     1
#define INITIAL   2
#define SHUTDOWN  3

/*****

int user_interface (int service)
{
    int ch, event;
    char parameter[6];

    switch ( service ) {
        case TEXT:  ch = keyboard_em(NEED, '\0');
                    return(ch);
        case EVENT: ch = keyboard_em(TRY, '\0');
                    if( ch == 0 )
                    {
                        ch = mouse_em(TRY, parameter);
                        if( ch == RLeftB )
                        {
                            event = window_em(COORD_TO_EVENT, parameter);
                            return(event);
                        }
                        return(NOEVENT);
                    }
                    event = keyboard_em(KEY_TO_EVENT, ch);
                    return(event);
        case INITIAL: ch = mouse_em(MINIT, "");
                    return(ch);
        case SHUTDOWN: ch = mouse_em(SYSOFF, "");
                    return(ch);
    } /* end switch( service ) */
    return(-1); /* no services provided if you get here */
}

```

```

/*.....*/
/* file: INIT.MGR */
/*.....*/

/* Initialize manager: The initialize manager sets up the agenda */
/* system. It loads the structure from file "AGM$.$$$" or create */
/* this file if it does not exist, and it sets up the month and */
/* day windows to the current system's date. */
/* NOTE: this manager is to be included in the system manager. */

/*.....*/
/* Initialize manager constant declaration */
/*.....*/

#define ON 1
#define OFF 0

/*.....*/

void initialize_manager (void)
{
    char param[12], *temp, password[10];

    mode = NORMAL;
    select = 1;
    into = 1;
    clrscr();
    string_em(INIT, "", "");
    if( !date_time_em(TODAY, param) ) error_em(TIME_MOD);
    temp = strtok(param, "/");
    strcpy(current_month, temp);
    temp = strtok(NULL, "/");
    strcpy(current_day, temp);
    temp = strtok(NULL, "/");
    strcpy(current_year, temp);
    if( !date_time_em(TIME, param) ) error_em(TIME_MOD);
    strcpy(current_time, param);
    temp = strtok(current_time, ":");
    strcpy(display_time, temp);
    strcat(display_time, ":");
    temp = strtok(NULL, ":");
    if( strcmp(temp, "30") >= 0 )
        strcat(display_time, "30");
    else
        strcat(display_time, "00");
    strcpy(param, current_month);
    strcat(param, "/");
    strcat(param, current_year);
    month_offset = date_time_em(FIRST_DAY, param);
    Agenda = NULL;
    cyear = NULL;
    cmonth = NULL;
    cday = NULL;
    cappointment = NULL;
    agenda_em(OPEN, "AGM$.$$$rt");
    if( (agenda_em(RETRIEVE, "")) && (strlen(ux) != 0) )
    {
        if( !window_em(OPEN_PASS, "") ) error_em(WINDOW_MOD);
        strcpy(param, "12.4.9");
        strcpy(password, "");
        string_em(EDIT, password, param);
        if( strcmp(password, ux) == 0 )
            level = 1;
        else
            if( strcmp(password, ex) == 0 )
                level = 2;
            else
                level = 3;
        window_em(CLOSE_WINDOW, "");
    }
    else
    {
        if( !window_em(OPEN_NOPASS, "") ) error_em(WINDOW_MOD);
        do {
            strcpy(param, "12.4.9");
            string_em(EDIT, ux, param);
        } while( strlen(ux) == 0 );
        window_em(WRITEIN, "1");
        do {
            strcpy(param, "12.4.9");
            string_em(EDIT, ex, param);
        } while( strlen(ex) == 0 );
        level = 1;
    }
}

```

APPENDIX A
Personal Agenda code listing
150

```
    window_em(CLOSE_WINDOW, "");
}
if( !window_em(INITIALIZE, "") ) error_em(WINDOW_MOD);
if( !window_em(FILLMONTH, "") ) error_em(WINDOW_MOD);
if( Agenda != NULL )
{
    window_em(SETAPP, "");
    agenda_em(SETVAR, "");
}
flash_day = 0;
if( !window_em(FILLDAY, "") ) error_em(WINDOW_MOD);
string_em(HIDE, "", "");
}
```


APPENDIX A
Personal Agenda code listing
151

```
/* *****  
/* file: CLOCK.MGR */  
/* *****  
/* Clock Manager: The clock manager update the running clock which */  
/* appears in the top right corner of the day window. */  
  
void update_clock (void)  
{  
    char temp[12];  
    if( mode == NORMAL )  
    {  
        date_time_em(TIME,temp);  
        if( strcmp(myclock,temp) != 0 )  
        {  
            window(37,1,76,25);  
            textattr(WHITE + (BLUE << 4));  
            gotoxy(32,2);  
            strcpy(myclock,temp);  
            cputs(myclock);  
            string_em(HIDE,"","");  
        }  
    }  
}
```

```

/* *****
/* file: NEWPASS.MGR */
/* *****

/* new password manager: The new password managers responds to the
/* NEW_PASS event. It allows the user to change the system passwords */

void new_password_manager (void)
{
    char new_user_password[10];
    char new_secretary_password[10];
    char param[7];

    if( !window_em(OPEN_NEW_PASS,"") ) error_em(WINDOW_MOD);
    strcpy(new_user_password,"");
    strcpy(param,"12,4,9");
    string_em(EDIT,new_user_password,param);
    if( strcmp(new_user_password, ux) == 0 )
    {
        window_em(WRITELN,"2");      /* write 'new user password' */
        strcpy(param,"12,4,9");
        string_em(EDIT,ux,param);
        window_em(WRITELN,"3");      /* write 'new secretary password' */
        strcpy(param,"12,4,9");
        strcpy(new_secretary_password,"");
        string_em(EDIT,new_secretary_password,param);
        if( strcmp(new_secretary_password, sx) != 0 )
            strcpy(sx, new_secretary_password);
    }
    if( !window_em(CLOSE_WINDOW,"") ) error_em(WINDOW_MOD);
    window_em(SETAPP,current_day);
    window_em(FILLDAY,current_day);
    string_em(HIDE,"","");
}

```

APPENDIX A
Personal Agenda code listing
153

```

                                /******
                                /* file: MESS.MGR */
                                /******
/* Message manager: the message manager responds to the MESSAGE event */
/* which is initiated when the any level user wants to edit the      */
/* message window.                                                    */

void message_manager(void)
{
    if( !window_em(OPEN_MESS,"") ) error_em(WINDOW_MOD);
    textattr(WHITE + (CYAN << 4) );
    string_em(EDIT_MESS,"","");
    window_em(CLOSE_WINDOW,"");
    textattr(LIGHTGRAY + (BLACK << 4));
    window(1,19,36,25);
    clrscr();
    window_em(SETAPP,current_day);
    agenda_em(SETVAR,"");
    window_em(FILLDAY,current_day);
    string_em(HIDE,"","");
}

```

APPENDIX A
Personal Agenda code listing
154

```

/* *****
/* file: SELDAY.MGR */
/* *****
/* Select Day Manager: The select day manager responds to the SELECT_DAY *
/* event which is activated by the clicking of the mouse button with the *
/* mouse cursor on one of the day square of the month window. *
/* NOTE: If the mouse cursor is on an empty square of the month window *
/* no action is taken. *
void select_day_manager (void)
{
    int x, y, mm, yy;

    if( mode == NORMAL )
    {
        if( (calendar_square > month_offset) &&
            (calendar_square <= (month_size + month_offset)) )
        {
            x = calendar_square - month_offset;
            itoa(x, current_day, 10);
            if( !agenda_em(SETVAR,"") ) error_em(AGENDA_MOD);
            if( !window_em(FILLDAY,"") ) error_em(WINDOW_MOD);
            string_em(HIDE,"","");
        }
    }
    else /* mode == PLANNING */
    {
        window_em(INITIALIZE,"");
        mode = NORMAL;
        if( into == 2)
        {
            month_offset = next_month_offset;
            month_size = next_month_size;
            mm = atoi(current_month) + 1;
            if( mm == 13 )
            {
                mm = 1;
                yy = atoi(current_year) + 1;
                itoa(yy, current_year, 10);
            }
            itoa(mm, current_month, 10);
        }
        if( !window_em(FILLMONTH,"") ) error_em(WINDOW_MOD);
        strcpy(myclock,""); /* make sure you display the clock anew */
        if( (calendar_square > month_offset) &&
            (calendar_square <= (month_size + month_offset)) )
        {
            x = calendar_square - month_offset;
            itoa(x, current_day, 10);
            if( !agenda_em(SETVAR,"") ) error_em(AGENDA_MOD);
            if( !window_em(FILLDAY,"") ) error_em(WINDOW_MOD);
            string_em(HIDE,"","");
        }
    }
}

```

APPENDIX A
Personal Agenda code listing
155

```

                                /******
                                /* file: NEXDAY.MGR */
                                /******
/* Next Day Manager: The next day manager responds to the NEXT_DAY */
/* event which is the activation of the right arrow key on the */
/* console. NOTE: If the current day is the last day of the month */
/* no action is taken. */
void next_day_manager (void)
{
    int x;

    x = atoi(current day);
    if( mode == NORMAL )
    {
        into = 1;
        select = 1;
        if( x < month_size )
        {
            x++;
            itoa(x, current day, 10);
            if( !agenda_em(SETVAR,"") ) error_em(AGENDA_MOD);
            if( !window_em(FILLDAY,"") ) error_em(WINDOW_MOD);
            string_em(HIDE,"","");
        }
    }
    else /* mode == PLANNING */
    {
        if( !window_em(FLASH,"0") ) error_em(WINDOW_MOD);
        switch( into ) {
            case 1: if( x < month_size )
                    {
                        x++;
                    }
                    else
                    {
                        x = 1;
                        into = 2;
                    }
                    break;
            case 2: if( x < next_month_size )
                    {
                        x++;
                    }
                    else
                    {
                        x = 1;
                        into = 1;
                    }
                    break;
        }
        itoa(x, current day, 10);
        flash day = atoi(current day);
        if( !window_em(FLASH,"1") ) error_em(WINDOW_MOD);
        string_em(HIDE,"","");
    }
}

```

APPENDIX A
Personal Agenda code listing
156

```

*****
* file: PREDAY.MGR *
*****
/* Previous Day Manager: The previous day manager responds to the *
/* PREVIOUS DAY event which is the activation of the left arrow *
/* key on the console. NOTE: If the current day is the first day *
/* of the month, no action is taken. */

void previous_day_manager (void)
{
    int x;
    x = atoi(current day);
    if( mode == NORMAL )
    {
        select = 1;
        into = 1;
        if( x > 1 )
        {
            x--;
            itoa(x, current day, 10);
            if( !agenda_em(SETVAR,"") ) error_em(AGENDA_MOD);
            if( !window_em(FILLDAY,"") ) error_em(WINDOW_MOD);
            string_em(HIDE,"","");
        }
    }
    else /* mode == PLANNING */
    {
        if( !window_em(FLASH,"0") ) error_em(WINDOW_MOD);
        switch( into ) {
            case 1: if( x > 1 )
                {
                    x--;
                    else
                    {
                        x = next_month_size;
                        into = 2;
                    }
                    break;
                }
            case 2: if( x > 1 )
                {
                    x--;
                    else
                    {
                        x = month_size;
                        into = 1;
                    }
                    break;
                }
        }
        itoa(x, current day, 10);
        flash day = atoi(current day);
        if( !window_em(FLASH,"1") ) error_em(WINDOW_MOD);
        string_em(HIDE,"","");
    }
}

```

APPENDIX A
Personal Agenda code listing
157

```

/* Month Up Manager: The month up manager responds to the MONTH UP */
/* event which is the activation of the page up key on the console */
/* or the mouse button clicking with the mouse cursor on the <NEXT> */
/* button in the month window. */

void month_up_manager (void)
{
    int im, iy, tempm, tempy;
    char param[12], mon[3], yea[5];

    im = atoi(current_month);
    if( im == 12 ) /* if end of the year */
    {
        iy = atoi(current_year);
        im = 1; /* goto january */
        iy++; /* of next year */
        itoa(iy, current_year, 10);
    }
    else im++;
    itoa(im, current_month, 10);
    if( mode == PLANNING )
    {
        tempm = im + 1;
        if( tempm == 13 )
        {
            tempm = 1;
            tempy = atoi(current_year) + 1;
        }
        else
        {
            tempy = atoi(current_year);
            itoa(tempm, param, 10);
            strcat(param, "/");
            itoa(tempy, yea, 10);
            strcat(param, yea);
            next_month_offset = date_time_em(FIRST_DAY, param);
        }
        strcpy(param, current_month);
        strcat(param, "/");
        strcat(param, current_year);
        month_offset = date_time_em(FIRST_DAY, param);
        if( !window_em(FILLMONTH, "") ) error_em(WINDOW_MOD);
        strcpy(current_day, "1");
        into = 1;
        if( mode == NORMAL )
        {
            if( !agenda_em(SETVAR, "") ) error_em(AGENDA_MOD);
        }
        flash_day = 1;
        if( !window_em(FLASH, "1") ) error_em(WINDOW_MOD);
        window_em(SETAPP, "");
        string_em(HIDE, "", "");
    }
}

```

```

/* Month Down Manager: The month up manager reponds to the MONTH_DOWN =
/* event which is the activation of the page down key on the console */
/* or the mouse button clicking with the mouse cursor on the <PREV> */
/* button in the month window. */

void month_down_manager (void)
{
    int im, iy, tempm, tempy;
    char param[12], yea[5];

    im = atoi(current_month);
    if( im == 1 ) /* if start of the year */
    {
        iy = atoi(current_year);
        im = 12; /* goto december */
        iy--; /* of previous year */
        itoa(iy, current_year, 10);
    }
    else
    {
        im--;
        if( mode == PLANNING )
            next_month_offset = month_offset;
        itoa(im, current_month, 10);
        strcpy(param, current_month);
        strcat(param, "/");
        strcat(param, current_year);
        month_offset = date_time_em(FIRST_DAY, param);
        if( !window_em(FILLMONTH, "") ) error_em(WINDOW_MOD);
        strcpy(current_day, "1");
        into = 1;
        if( mode == NORMAL )
        {
            if( !agenda_em(SETVAR, "") ) error_em(AGENDA_MOD);
        }
        flash day = 1;
        if( !window_em(FLASH, "1") ) error_em(WINDOW_MOD);
        window_em(SETAPP, "");
        string_em(HIDE, "", "");
    }
}

```


APPENDIX A
Personal Agenda code listing
159

```
/* *****  
/* file: TWOM.MGR */  
/* *****  
  
/* Two Month manager: The two month manager responds to the PLAN */  
/* event. It changes the display to a two consecutive month windows */  
/* allowing the user to see appointment planning for these months. */  
/* When this manager is called the system enters a new state called */  
/* the planning state. */  
  
void two_month_manager (void)  
{  
    int im, iy;  
    char yea[5], param[12];  
  
    if( !window_em(TWO_MONTH,"") ) error_em(WINDOW_MOD);  
    mode = PLANNING;  
    select = 1;  
    into = 1;  
    im = atoi(current_month) + 1;  
    if( im == 13 )  
    {  
        im = 1;  
        iy = atoi(current_year) + 1;  
        itoa(iy, yea, 10);  
    }  
    else  
        strcpy(yea, current_year);  
    itoa(im, param, 10);  
    strcat(param, "/");  
    strcat(param, yea);  
    next month offset = date time em(FIRST_DAY, param);  
    if( !window_em(FILLMONTH,"") ) error_em(WINDOW_MOD);  
    if( !window_em(FLASH, "1") ) error_em(WINDOW_MOD);  
    if( !window_em(SETAPP,"") ) error_em(WINDOW_MOD);  
    string_em(HIDE,"","");  
}
```

APPENDIX A
Personal Agenda code listing
160

```

*****
* file: SCRUP.MGR *
*****

/* Scroll up Manager: The scroll down manager reponds to the
/* SCROLL_UP event which is the activation of the up arrow key
/* on the console or the clicking of the mouse button with the
/* mouse cursor on the up pointer in the day window.
/* NOTE: Time is circular on the current day (i.e. before 00 30
/* comes 00:00 then 23:30 on the same current day).

void scroll_up_manager (void)
{
    int ho, min;
    char *temp, c[3];

    temp = strtok(display_time, ":");
    ho = atoi(temp);
    temp = strtok(NULL, ":");
    min = atoi(temp);
    if( min == 0 )
    {
        min = 30;
        ho--;
    }
    else
        min = 0;
    if( ho == -1 )
        ho = 23;
    itoa(ho, c, 10);
    if( ho < 10 )
    {
        strcpy(display_time, "0");
        strcat(display_time, c);
    }
    else
        strcpy(display_time, c);
    strcat(display_time, ":");
    if( min == 0 )
        strcat(display_time, "00");
    else
        strcat(display_time, "30");
    if( !window_em(FILLDAY, "") ) error_em(WINDOW_MOD);
    string_em(HIDE, "", "");
}

```

APPENDIX A
Personal Agenda code listing
161

```

/* *****
/* file: SCROLL.MGR */
/* *****

/* Scroll Down Manager: The scroll down manager responds to the */
/* SCROLL_DOWN event which is the activation of the down arrow key */
/* on the console or the clicking of the mouse button with the */
/* mouse cursor on the down pointer in the day window. */
/* NOTE: Time is circular on the current day (i.e. after 23:30 */
/* comes 00:00 then 00:30 on the same current day). */

void scroll_down_manager (void)
{
    int ho, min;
    char *temp, c[3];

    temp = strtok(display_time, ":");
    ho = atoi(temp);
    temp = strtok(NULL, ":");
    min = atoi(temp);
    if( min == 0 )
        min += 30;
    else
    {
        min = 0;
        ho++;
    }
    if( ho == 24 )
        ho = 0;
    itoa(ho, c, 10);
    if( ho < 10 )
    {
        strcpy(display_time, "0");
        strcat(display_time, c);
    }
    else
        strcpy(display_time, c);
    strcat(display_time, ":");
    if( min == 0 )
        strcat(display_time, "00" );
    else
        strcat(display_time, "30" );
    if( !window_em(FULLDAY, "") ) error_em(WINDOW_MOD);
    string_em(HIDE, "", "");
}

```

```

/* *****
/* file: ADD.MGR */
/* *****

/* Add manager: The add manager manages the ADD event responses *
/* An add window is open and user interaction is monitored.      *
/* This manager is to be included in the system manager.         */

void add_manager (void)
{
    char start[6] = "", endt[6] = "", comment[36] = "", param[80];
    int event;

    if( !window_em(OPEN_ADD,"") ) error_em(WINDOW_MOD);
    string_em(HIDE,"","");
    do {
        event = system_manager(WTE_inet);
        window(5,6,75,15);
        switch ( event ) {
            case NEXT_BOX: current_box++;
                           if( current_box == 4 )
                               current_box = 1;
                           if( !window_em(HIGHLIGHT,"") )
                               error_em(WINDOW_MOD);
                           string_em(HIDE,"","");
                           break;
            case ACTIVATE: switch ( current_box ) {
                           case 1: strcpy(param,"8,5,5");
                               string_em( EDIT, start, param);
                               break;
                           case 2: strcpy(param,"17,5,5");
                               string_em( EDIT, endt, param);
                               break;
                           case 3: strcpy(param,"25,5,36");
                               string_em( EDIT, comment, param);
                               break;
                           }
                           break;
            case START_TIME: if( current_box != 1 )
                            {
                                current_box = 1;
                                if( !window_em(HIGHLIGHT,"") )
                                    error_em(WINDOW_MOD);
                                strcpy(param,"8,5,5");
                                string_em( EDIT, start, param);
                                break;
                            }
            case END_TIME: if( current_box != 2 )
                            {
                                current_box = 2;
                                if( !window_em(HIGHLIGHT,"") )
                                    error_em(WINDOW_MOD);
                                strcpy(param,"17,5,5");
                                string_em( EDIT, endt, param);
                                break;
                            }
            case COMMENT: if( current_box != 3 )
                            {
                                current_box = 3;
                                if( !window_em(HIGHLIGHT,"") )
                                    error_em(WINDOW_MOD);
                                strcpy(param,"25,5,36");
                                string_em( EDIT, comment, param);
                                break;
                            }
            case PRIVATE:
            case OK: if( (strlen(start') && (strlen(endt))) )
                    {
                        strcpy(param, start);
                        strcat(param, ",");
                        strcat(param, endt);
                        strcat(param, ",");
                        if( event == PRIVATE )
                            strcat(param, "1");
                        else
                            strcat(param, "0");
                        strcat(param, ",");
                        strcat(param, comment);
                        if( agenda_em(INSERT, param) == TIME_BOOKED)
                            error_em(TIME_CONFLICT);
                        window(5,6,75,15);
                        string_em(HIDE,"","");
                    }
        }
    } while( event != QUIT );
}

```

APPENDIX A
Personal Agenda code listing
163

```
        if( !window_em(FILLDAY,"") ) error_em(WINDOW_MOD);
    }
    else
    {
        error_em(INCOMPLETE);
        window(5,6,75,15);
        event = 0;
        string_em(HIDE,"","");
    }
    break;
    case CANCEL: break;
} /* end switch ( event ) */
} while( (event != OK) && (event != CANCEL) && (event != PRIVATE) );
if( !window_em(CLOSE_WINDOW,"") ) error_em(WINDOW_MOD);
window_em(SETAPP,current_day);
agenda_em(SETVAR,"");
window_em(FILLDAY,current_day);
string_em(HIDE,"","");
}
```

```

.....
/* file: DEL.MGR */
.....

/* Delete manager: The delete manager manages the DELETE event *
/* responses. A Delete window is open and user interaction is *
/* monitored. */

void delete_manager (void)
{
    char param[80], start[6] = "", endt[6] = "",
    int event;

    if( !window_em(OPEN_DELETE,"") ) error_em(WINDOW_MOD);
    string_em(HIDE,"","");
    do {
        event = system_manager(WTE_inst);
        window(5,6,75,15);
        switch ( event ) {

            case NEXT_BOX: current_box++;
                if( current_box == 4 )
                    current_box = 1;
                if( !window_em(HIGHLIGHT,"") ) error_em(WINDOW_MOD);
                string_em(HIDE,"","");
                break;

            case ACTIVATE: switch ( current_box ) {
                case 1: strcpy(param,"8,5,5");
                    string_em( EDIT, start, param);
                    if( strlen(start) == 0 )
                        break;
                    if( agenda_em(MATCH_ST,start))
                        window_em(FILL_COH,"");
                    else
                    {
                        error_em(NO_MATCH);
                        window(5,6,75,15);
                        string_em(HIDE,"","");
                    }
                    break;
                case 2: strcpy(param,"17,5,5");
                    string_em( EDIT, endt, param);
                    if( strlen(endt) == 0 )
                        break;
                    if( agenda_em(MATCH_ET,endt))
                        window_em(FILL_COH,"");
                    else
                    {
                        error_em(NO_MATCH);
                        window(5,6,75,15);
                        string_em(HIDE,"","");
                    }
                    break;
            }
            break;

            case START_TIME: if( current_box != 1 )
                {
                    current_box = 1;
                    if( !window_em(HIGHLIGHT,"") )
                        error_em(WINDOW_MOD);
                }
                strcpy(param,"8,5,5");
                string_em( EDIT, start, param);
                if( strlen(start) == 0 )
                    break;
                if( agenda_em(MATCH_ST,param))
                    window_em(FILL_COH,"");
                else
                {
                    error_em(NO_MATCH);
                    window(5,6,75,15);
                    string_em(HIDE,"","");
                }
                break;

            case END_TIME: if( current_box != 2 )
                {
                    current_box = 2;
                    if( !window_em(HIGHLIGHT,"") )
                        error_em(WINDOW_MOD);
                }
                strcpy(param,"17,5,5");

```

APPENDIX A
Personal Agenda code listing
165

```
string_em( EDIT, endt, param);
if( strlen(endt) == 0 )
    break;
if( agenda_em(MATCH_ET,param) )
    window_em(FILL_COM,"");
else
{
    error_em(NO MATCH);
    window(5,6,75,15);
    string_em(HIDE,"","");
}
break;

case OK: if( cappointment != NULL )
{
    if( !agenda_em(ADELETE, start) )
        error_em(GENDA_MOD);
}
break;

case CANCEL: break;
default: break;
} /* end of loop ( event ) */
} while( event != OK) && (event != CANCEL) );
if( !window_em(CLOSE WINDOW,"") ) error_em(WINDOW_MOD);
window_em(SETAPP,current_day);
agenda_em(SETVAR,"");
window_em(FILLDAY,current_day);
string_em(HIDE,"","");
}
```

```

/* *****
 * file: MOD.MGR *
 * ***** */

/* Modify manager: The delete manager manages the MODIFY event *
/* responses. A Modify window is open and user interaction is *
/* monitored. At the first OK received the appointment search *
/* is terminated and the user can modify the found appointment. *
/* A second OK command is needed for the modifications to take *
/* effect, note that if the changes conflict with an existing *
/* appointment the old data is re-entered. */

void modify_manager (void)
{
    char param[80], old_start[6] = "", old_end[6] = "", old_comment[36];
    char new_start[6], new_end[6], new_comment[36], cpriv[3];
    int event, flag, priv;

    if( !window_em(OPEN MODIFY,"") ) error_em(WINDOW_MOD);
    string_em(HIDE,"","");
    do {
        event = system_manager(WTE_inst);
        window(5,6,75,15);
        switch ( event ) {

            case NEXT_BOX: current_box++;
                          if( current_box == 4 )
                              current_box = 1;
                          if( !window_em(HIGHLIGHT,"") )
                              error_em(WINDOW_MOD);
                          string_em(HIDE,"","");
                          break;

            case ACTIVATE: switch ( current_box ) {
                          case 1: strcpy(param,"8,5,5");
                              string_em( EDIT, old_start, param);
                              if( strlen(old_start) == 0 )
                                  break;
                              if( agenda_em(MATCH ST,old_start))
                                  window_em(FILL_COH,"");
                              else
                                  {
                                      error_em(NO MATCH);
                                      window(5,6,75,15);
                                      string_em(HIDE,"","");
                                  }
                              break;
                          case 2: strcpy(param,"17,5,5");
                              string_em( EDIT, old_end, param);
                              if( strlen(old_end) == 0 )
                                  break;
                              if( agenda_em(MATCH ET,old_end))
                                  window_em(FILL_COH,"");
                              else
                                  {
                                      error_em(NO MATCH);
                                      window(5,6,75,15);
                                      string_em(HIDE,"","");
                                  }
                              break;
                          }
                          break;

            case START_TIME: if( current_box != 1 )
                          {
                              current_box = 1;
                              if( !window_em(HIGHLIGHT,"") )
                                  error_em(WINDOW_MOD);
                          }
                          strcpy(param,"8,5,5");
                          string_em( EDIT, old_start, param);
                          if( strlen(old_start) == 0 )
                              break;
                          if( agenda_em(MATCH ST,old_start))
                              window_em(FILL_COH,"");
                          else
                              {
                                  error_em(NO MATCH);
                                  window(5,6,75,15);
                                  string_em(HIDE,"","");
                              }
                          break;

            case END_TIME: if( current_box != 2 )

```



```

        {
            current_box = 2;
            if( !window_em(HIGHLIGHT,"") )
                error_em(WINDOW_MOD);
        }
        strcpy(param,"17,5,5");
        string_em( EDIT, old_end, param);
        if( strlen(old_end) == 0 )
            break;
        if( agenda_em(MATCH ET,old_end))
            window_em(FILL_CON,"");
        else
        {
            error_em(NO MATCH);
            window(5,6,75,15);
            string_em(HIDE,"","");
        }
        break;
case OK: if( cappointment != NULL )
{
    strcpy(old_start, cappointment->start_time);
    strcpy(old_end, cappointment->end_time);
    strcpy(old_comment, cappointment->comment);
    strcpy(new_start, cappointment->start_time);
    strcpy(new_end, cappointment->end_time);
    strcpy(new_comment, cappointment->comment);
    priv = cappointment->private;
    flag = 0; /* flag any changes done to appointment */
    do {
        event = system_manager(WTE_inst);
        switch ( event ) {

            case NEXT_BOX: current_box++;
                            if( current_box == 4 )
                                current_box = 1;
                            if( !window_em(HIGHLIGHT,"") )
                                error_em(WINDOW_MOD);
                            string_em(HIDE,"","");
                            break;

            case ACTIVATE: switch ( current_box ) {
                            case 1: strcpy(param,"8,5,5");
                                    string_em( EDIT, new_start, param);
                                    flag = 1;
                                    break;
                            case 2: strcpy(param,"17,5,5");
                                    string_em( EDIT, new_end, param);
                                    flag = 1;
                                    break;
                            case 3: strcpy(param,"25,5,36");
                                    string_em(EDIT, new_comment, param);
                                    flag = 1;
                                    break;
                            }
                            break;

            case START_TIME: if( current_box != 1 )
                            {
                                current_box = 1;
                                if( !window_em(HIGHLIGHT,"") )
                                    error_em(WINDOW_MOD);
                            }
                            strcpy(param,"8,5,5");
                            string_em( EDIT, new_start, param);
                            flag = 1;
                            break;

            case END_TIME: if( current_box != 2 )
                            {
                                current_box = 2;
                                if( !window_em(HIGHLIGHT,"") )
                                    error_em(WINDOW_MOD);
                            }
                            strcpy(param,"17,5,5");
                            string_em( EDIT, new_end, param);
                            flag = 1;
                            break;

            case COMMENT: if( current_box != 3 )
                            {
                                current_box = 3;
                                if( !window_em(HIGHLIGHT,"") )
                                    error_em(WINDOW_MOD);
                            }

```

APPENDIX A
Personal Agenda code listing
168

```

    }
    strcpy(param,"25.5.36");
    string_em(EDIT,new_comment,param);
    flag = 1;
    break;

case OK: if( flag )
    {
        if( (strlen(new_start) == 0) ||
            (strlen(new_end) == 0) )
            break;
        if( !agenda_em(ADELETE,old_start) )
            error_em(AGENDA_MOD);
        strcpy(param,new_start);
        strcat(param,".");
        strcat(param,new_end);
        strcat(param,".");
        itoa(priv,cpriv,10);
        strcat(param,cpriv);
        strcat(param,".");
        strcat(param,new_comment);
        if( agenda_em(INSERT,param) == TIME_BOOKED )
        {
            /** if insert new data time conflict re-insert old */
            error_em(TIME_CONFLICT);
            strcpy(param,old_start);
            strcat(param,".");
            strcat(param,old_end);
            strcat(param,".");
            itoa(priv,cpriv,10);
            strcat(param,cpriv);
            strcat(param,".");
            strcat(param,old_comment);
            if( !agenda_em(INSERT,param) )
                error_em(AGENDA_MOD);
        }
    }
    break; /* case OK */

case CANCEL: break;
default: break;
} /* end switch( event ) number 2 */
} while( (event != OK) && (event != CANCEL) );
} /* end if( cappointment != NULL ) */
break;

case CANCEL: break;
default: break;
} /* end switch( event ) */
} while( (event != OK) && (event != CANCEL) );
if( !window_em(CLOSE_WINDOW,"") ) error_em(WINDOW_MOD);
window_em(SETAPP,current_day);
agenda_em(SETVAR,"");
window_em(FILLDAY,current_day);
string_em(HIDE,"","");
}

```

APPENDIX A
Personal Agenda code listing
169

```

/* *****
/* file: QUIT.MGR */
/* *****

/* Quit manager: The quit manager responds to the 'Quit' command */
/* It saves the agenda structure into file "AG##.$$$". */

void quit_manager (void)
{
    window(1,1,80,25);
    textattr(LIGHTGRAY + (BLACK << 4));
    clrscr();
    if( !agenda_em(SAVE,"ag##.$$$") ) error_em(AGENDA_MOD);
    if( !agenda_em(CLOSE,"") ) error_em(AGENDA_MOD);
}

```

```

/* *****
 * file: MOUSE.EM */
/* *****

/* Mouse Execution Module: The mouse EM returns mouse input to the
 * event manager. It provides two services: initialize (INIT) and
 * try for input (TRY). INIT is straight forward it checks to see
 * if mouse is present and initializes the proper parameters. The
 * TRY service returns the mouse action ch value as defined in the
 * file CHAR.H. When a mouse button is depressed, the parameter
 * string is set to the x and y coordinate of the mouse cursor. Of
 * course the coordinate are converted into characters before.

#include <DOS.H>
#include <STDLIB.H>
#include <STRING.H>
#include "char.h"

/* *****
 * Services provided by MOUSE.EM */
/* *****

#define MINIT 0
#define TRY 1
#define SYSOFF 2

/* *****
 * Global variable needed by mouse services */
/* *****

struct REGPACK *preg; /* register structure for interrupt calls */

/* *****
 * mouse_installed: Check and return true if the mouse is installed
 * on the system. */
int mouse_installed (void)
{
    if( peek(0x0000,0x00cc) ) /* if interrupt vector not 0 */
    {
        preg->r_ax = 0;
        intr(0x33,preg);
        return( preg->r_ax != 0 ); /* if ax = 0 mouse not installed */
    }
    else
        return(0); /* interrupt vector is 0 then no mouse */
}

/* *****
 * show_mouse_cursor: Shows the mouse cursor. */
void show_mouse_cursor (void)
{
    preg->r_ax = 1;
    intr(0x33,preg);
}

/* *****
 * hide_mouse_cursor: Hides the mouse cursor. */
void hide_mouse_cursor (void)
{
    preg->r_ax = 2;
    intr(0x33,preg);
}

/* *****
 * move_mouse: moves the mouse cursor to x and y coordinates */
void move_mouse ( int xcoord, int ycoord )
{
    preg->r_ax = 4;
    preg->r_cx = (xcoord * 8) - 1;
    preg->r_dx = (ycoord * 8) - 1;
    intr(0x33,preg);
}

/* *****
 * get_mouse_action: mouse action is returned through parameters */
void get_mouse_action ( int *but, int *hor, int *ver )
{
    div_t x;

```

```

preg->r_ax = 3;
intr(0x33,preg);
x = div( preg->r_cx, 8 );
*hor = x.quot;
x = div( preg->r_dx, 8 );
*ver = x.quot;
if( ( (preg->r_bx & 0x1) != 0x1 ) && ( (preg->r_bx & 0x2) != 0x2 ) )
{
    *but = NoB;
    return;
}
if( ( (preg->r_bx & 0x1) == 0x1 ) && ( (preg->r_bx & 0x2) == 0x2 ) )
    *but = BothB;
else
    if( ( (preg->r_bx & 0x1) == 0x1 ) )
        *but = LeftB;
    else *but = RightB;
}

/*****/
/* set_mouse_cursor: set the mouse cursor size and shape. */
void set_mouse_cursor ( int Ordchar )
{
    preg->r_ax = 10;
    preg->r_bx = 0; /* software text cursor */
    preg->r_cx = 0x7700;
    preg->r_dx = Ordchar & 0x77;
    intr(0x33,preg);
}

/*****/
/* mouse_pressed: returns true if mouse 'button' is pressed. */
int mouse_pressed ( int button )
{
    preg->r_ax = 5;
    preg->r_bx = button;
    intr(0x33,preg);
    return( preg->r_bx > 0 );
}

/*****/
/* mouse_released: returns true when mouse 'button' is released */
int mouse_released ( int button )
{
    preg->r_ax = 6;
    preg->r_bx = button;
    intr(0x33,preg);
    return( preg->r_bx > 0 );
}

/*****/

                                /*****/
                                /* MOUSE EXECUTION MODULE */
                                /******/

int mouse_em ( int service, char *parameter )
{
    static int cx, cy; /* mouse home coordinate, set by system manager */
    int Hor, Ver; /* mouse movement coordinate for click action */
    int ch, B;
    char temp[3];

    switch( service ) {
        case MINIT: if( mouse_installed() )
            {
                cx = 40;
                cy = 13;
                move_mouse( cx, cy );
                show_mouse_cursor();
                return( SET );
            }
            return( RESET );

        case SYSOFF: if( mouse_installed() )
            {
                hide_mouse_cursor();
                return(SET);
            }

        case TRY: get_mouse_action( &B, &Hor, &Ver );
    }
}

```

```

switch ( B ) {
  case LeftB: ch = MLeftB;
               while( !mouse_released(0) );
               break;

  case RightB: ch = MRightB;
               while( !mouse_released(1) );
               break;

  default: if( (Ver - cy) > 1 )
            ch = MDown;
            else
            if( (cy - Ver) > 1 )
            ch = MUp;
            else
            if( (Hor - cx) > 1 )
            ch = MRight;
            else
            if( (cx - Hor) > 1 )
            ch = MRight;
            break; /* default */
} /* end switch ( B ) */
if ( (B == LeftB) || (B == RightB) )
{
  itoa(Hor,parameter,10);
  strcat(parameter,",");
  itoa(Ver,temp,10);
  strcat(parameter,temp);
}
} /* end switch ( service ) */
return(ch);
}

```

```

/*-----*/
/* file: KEYBOAR.EM */
/*-----*/

/* Keyboard Execution Module: The Keyboard EM provides two services to
/* the system manager: try input (TRY) and need input (NEED). The
/* NEED service will wait for an input on the keyboard, while the TRY
/* will return -1 if no input is present. Characters returned by the
/* keyboard EM are those defined in the file CHAR.H and the normal
/* ASCII code for the alphabet letters and numbers. */

#include <DOS.H>
#include <STDLIB.H>
#include <CONIO.H>
#include "char.h"
#include "event.h"

/*-----*/
/* Services provided by the keyboard EM */
/*-----*/

#define NEED      0
#define TRY      1
#define KEY_TO_EVENT 2
#define NO_SERVICE -1

/*-----*/
/* KEYBOARD EXECUTION MODULE */
/*-----*/

int keyboard_em (int service, int key)
{
    int B, ch, getch(), mm, 1/;

    switch ( service ) {
        case NEED: ch = getch();
            if( ch == '\0' )
            {
                ch = getch();
                if( ( (ch >= 15) && (ch <= 25) ) ||
                    ( (ch >= 30) && (ch <= 38) ) ||
                    ( (ch >= 44) && (ch <= 50) ) ||
                    ( (ch >= 59) && (ch <= 68) ) ||
                    ( (ch >= 71) && (ch <= 73) ) ||
                    ( (ch >= 75) && (ch <= 77) ) ||
                    ( (ch >= 77) && (ch <= 127) ) )
                {
                    ch += 128;
                }
                else
                {
                    if( (ch >= 128) && (ch <= 140) )
                    {
                        ch += 6;
                    }
                    else return(0); /* not a valid key combination */
                }
                /* end if ( ch == '\0' ) */
                return(ch);
            }
        case TRY: if( kbhit() )
            {
                ch = getch();
                if( ch == '\0' )
                {
                    ch = getch();
                    if( ( (ch >= 15) && (ch <= 25) ) ||
                        ( (ch >= 30) && (ch <= 38) ) ||
                        ( (ch >= 44) && (ch <= 50) ) ||
                        ( (ch >= 59) && (ch <= 68) ) ||
                        ( (ch >= 71) && (ch <= 73) ) ||
                        ( (ch >= 75) && (ch <= 77) ) ||
                        ( (ch >= 77) && (ch <= 127) ) )
                    {
                        ch += 128;
                    }
                    else
                    {
                        if( (ch >= 128) && (ch <= 140) )
                        {
                            ch += 6;
                        }
                        else return(0);
                    }
                    /* end if ( ch == '\0' ) */
                    return(ch);
                }
                /* end if ( kbhit() ) */
                return(0); /* not kbhit() */
            }
        case KEY_TO_EVENT: if(window_number == 0) /* don't look at these when */
            switch( key ) { /* in a function window */
                case PgUp: return(MONTH_UP);
                case PgDn: return(MONTH_DOWN);
                case Rarr: return(NEXT_DAY);
                case Larr: return(PREVIOUS_DAY);
            }
    }
}

```

APPENDIX A
Personal Agenda code listing
174

```

case Uarr: if( mode == NORMAL )
            return(SCROLL_UP);
            return(NOEVENT);
case Darr: if( mode == NORMAL )
            return(SCROLL_DOWN);
            return(NOEVENT);
case 'A':
case 'a': if( mode == NORMAL )
            return(ADD);
            return(NOEVENT);
case 'D':
case 'd': if( mode == NORMAL )
            return(DELETE);
            return(NOEVENT);
case 'M':
case 'm': if( mode == NORMAL )
            return(MODIFY);
            return(NOEVENT);
case 'P':
case 'p': if( mode == NORMAL )
            {
                if( level == 1 )
                    return(PASSWORD);
            }
            return(NOEVENT);
case 'S':
case 's': if( mode == NORMAL )
            return(MESSAGE);
            return(NOEVENT);
case 'L':
case 'l': if( mode == NORMAL )
            return(PLAN);
            return(NOEVENT);
case Enter: if( mode == PLANNING )
            {
                if( into == 1 )
                    calendar_square = atoi(current_day) + month_offset;
                else /* select == 2 */
                    calendar_square = atoi(current_day) + next_month_offset;
                return(SELECT_DAY);
            }
            return(NOEVENT);
case 'Q':
case 'q': if( mode == NORMAL )
            return(QUIT);
            return(NOEVENT);
} /* end first switch */
else /* window number > 0 */
switch( key ) {
    case Tab: return(NEXT_BOX);
    case Ins: return(ACTIVATE);
    case Esc: return(CANCEL);
    case Enter: return(OK);
    case AltP: return(PRIVATE);
    default: return(NOEVENT);
} /* end second switch ( key ) */
default: return(NO_SERVICE); /* no service provided */
} /* end switch ( service ) */
}

```



```

/* *****
/* file: TIME.EM */
/* *****
/* Time Execution Module: The time EM contains the time functions and */
/* procedures. The Julian date is very usefull when calculating first */
/* day of the month (i.e. is first day a friday, monday etc.). */
/* julday and caldat where taken from the book "Numerical Recipes the */
/* Art of Scientific Computing" by W. Press, B. Flannery, S. Teukolsky */
/* and W. Vetterling, Cambridge University Press, Cambridge, New York */
/* 1986. This book is for PASCAL, conversion to C was quite difficult */
#include <STDLIB.H>
#include <STRING.H>
#include <DOS.H>
#include <TIME.H>

/* *****
/* Services provided by date/time execution module */
/* *****
#define JULIAN      0
#define CALDAT      1
#define TODAY       2
#define TIME        3
#define COUNTER      4
#define FIRST DAY 5

/* *****
/* Date/time execution module constants definitions */
/* *****
#define ERROR      0
#define Igreg 588829 /* julian = Oct.15 1582 Georgian calendar adopted */
#define Igreg2 2299161 /* correction for cross-over to Georgian calendar */
#define SEPARATOR ".../" /* parameters separators */
/* *****
/* julday: returns the julian date for day id, month mm, and year iyyy */
long julday ( int mm, int id, int iyyy )
{
    /* local variables */
    long ja, jm, jy, jul, temp;

    if( iyyy == 0 ) return(ERROR);
    if( iyyy < 0 ) iyyy++;
    if( mm > 2 )
    {
        jy = iyyy;
        jm = mm + 1;
    }
    else
    {
        jy = iyyy - 1;
        jm = mm + 13;
    }
    jul = (365.25 * jy);
    jul = jul + (30.6001 * jm) + id + 1720995;
    temp = (long)(id) + 31 * ((long)(mm) + 12 * (long)(iyyy));
    if ( temp >= Igreg )
    {
        ja = 0.01 * jy;
        jul = jul + 2 - ja + ( 0.25 * ja );
    }
    return( jul );
}

/* *****
/* caldat: returns the Gregorian date for the julian date 'julian'. The */
/* Gregorian date is returned through the parameters mm, id, and iyyy */
void caldat ( long julian, int *mm, int *id, int *iyyy )
{
    /* local variables */
    long je, jd, jc, jb, ja, jalpha;
    int temp;

    if( julian >= Igreg2 )
    {
        jalpha = ((julian - 1867216) - 0.25) / 36524.25;
        temp = 0.25 * jalpha;
        ja = julian + 1 + jalpha - temp;
    }
    else
    {
        ja = julian;
        jb = ja + 1524;
        jc = 6680 + (((jb - 2439870) - 122.1) / 365.25);
        jd = 365.25 * jc;
        je = (jb - jd)/30.6001;
        temp = 30.6001 * je;
    }
}

```

APPENDIX A
Personal Agenda code listing
176

```

*id = jb - jd - temp;
*mm = je - 1;
if( *mm > 12 ) *mm -= 12;
*yyyy = jc - 4715;
if( *mm > 2 ) *yyyy -= 1;
if( *yyyy <= 0 ) *yyyy += 1;
}

/*****
/* system_time: Returns the systems time in hour:minute. Time is returned*/
/* through the parameter time_now.
void system_time (char *time_now)
{
    struct time now;
    char temp[5];

    gettimeofday(&now);
    itoa(now.tv_hour,time_now,10);
    strcat(time_now,":");
    itoa(now.tv_min,temp,10);
    if( now.tv_min < 10 )
        strcat(time_now,"0");
    strcat(time_now,temp);
}

/*****
/* system_date: Returns the systems date in month/day/year. Date is
/* returned through the parameter today.
void system_date ( char *today )
{
    struct date now;
    char temp[5];

    getdate(&now);
    itoa(now.da_mon,today,10);
    strcat(today,"/");
    itoa(now.da_day,temp,10);
    strcat(today,temp);
    strcat(today,"/");
    itoa(now.ca_year,temp,10);
    strcat(today,temp);
}

/*****
/* timer_value: Returns a character string representing the long integer */
/* value of the time in seconds, elapsed since 00:00:00 GMT, January 1, */
/* 1970.
void timer_value ( char *val )
{
    time_t now;

    time(&now);
    ltoa(now,val,10);
}

/*****
/* mod: returns the value of x modulo y
int mod ( long x, int y )
{
    int ret;

    while( (x = x - y) >= y );
    ret = x;
    return(ret);
}

/*****
/* Date/Time execution module code */
/*****

int date_time_em ( int service, char *parameters )
{
    int id, mm, yyyy;
    long jul;
    char temp[33], *ptr;

    switch ( service ) {
        case JULIAN: ptr = strtok( parameters, SEPARATOR );
                    mm = atoi(ptr);
                    if( !mm ) return(0);
                    ptr = strtok( NULL, SEPARATOR );

```

```

    id = atoi(ptr);
    if( !id ) return(0);
    ptr = strtok( NULL, SEPARATOR );
    yyyy = atoi(ptr);
    if( !yyyy ) return(0);
    jul = julday( mm, id, yyyy );
    ltoa( jul, parameters, 10 );
    return(1);
case CALDAT: jul = atoi( parameters );
    caldat( jul, &mm, &id, &yyyy );
    ltoa( mm, parameters, 10 );
    strcat( parameters, "/" );
    ltoa( id, temp, 10 );
    strcat( parameters, temp );
    strcat( parameters, "/" );
    ltoa( yyyy, temp, 10 );
    strcat( parameters, temp );
    return(1);
case TODAY: system date( parameters );
    return(1);
case TIME:  system time( parameters );
    return(1);
case COUNTER: timer value( parameters );
    return(1);
case FIRST_DAY: ptr = strtok( parameters, SEPARATOR );
    mm = atoi(ptr);
    if( !mm ) return(0);
    ptr = strtok( NULL, SEPARATOR );
    yyyy = atoi(ptr);
    if( !yyyy ) return(0);
    id = 1;
    jul = julday( mm, id, yyyy );
    jul++;
    jul -= 21000000; /* reduce long to speed modulo */
    id = mod(jul,7);
    return(id); /* value of id 0 = sunday ,6 = saturday */
} /* end switch ( service ) */

```

```

.....
* STRING.EM *
.....

/* String execution module: The string EM contains all the procedures *
 * and functions for the cursor handling and character string editing *

/* NOTE: Most of these procedures/functions are from Borland TURBO C *
 * MCCALC.C example provided with the TURBO C Professional *
 * Package. *

#include <stdio.h>
#include <string.h>
#include <dos.h>
#include <conio.h>
#include <mem.h>
#include "char.h"

.....
/* String EM services */
.....

#define EDIT      1
#define INIT      2
#define HIDE      3
#define EDIT_MESS 4

.....
/* Execution module private variable */
.....

static int Tallcursor, Shortcursor;
char temp[80];

.....
/* String EM constant */
.....

#define SEPARATOR1 ".../"
#define legal " ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz\
0123456789_$%.,:~*#!?+/-/(){}[]!@<>="

.....
/* setcursor: DOS function to set the character cursor */
void setcursor ( unsigned int shape )
{
    union REGS reg;

    reg.h.ah = 1;
    reg.x.cx = shape;
    int86(0x10, &reg, &reg);
}

.....
/* hidecursor: Used to hide the text cursor */
void hidecursor(void)
{
    union REGS reg;

    reg.h.ah = 2;
    reg.h.bh = 0;
    reg.x.dx = 0x1900;
    int86(0x10, &reg, &reg);
}

.....
/* changecursor: Changes the cursor from insert mode to overwrite */
/* mode or vice versa according to inmode value */
void changecursor ( int inmode )
{
    if( inmode )
        setcursor(Tallcursor);
    else
        setcursor(Shortcursor);
}

.....
/* initcursor: Initialize the short and tall cursor value */
void initcursor (void)

```

3 of/de 3

MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS
STANDARD REFERENCE MATERIAL 1010a
(ANSI and ISO TEST CHART No. 2)

```

{
    struct text_info ti;

    gettextinfo(&ti);
    if( ti.currmode == MONO )
    {
        Shortcursor = 0X0A0C;
        Tallcursor = 0X090B;
    }
    else
    {
        Shortcursor = 0X0607;
        Tallcursor = 0X0007;
    }
    setcursor(Shortcursor);          /* set cursor off */
    hidecursor();
}

/*****
/* editstring: This function permits editing of string s.
/* To terminate editing <enter> or <esc> can be used. If <enter>
/* is used the changes are inserted for good, if <esc> is pressed the
/* changes are aborted and s is returned to its original state
*/
int editstring (char s[], int posx, int posy, int maxlength)
{
    int c, i, len = strlen(s), pos = len, insert = 1;
    char ch;

    textattr(LIGHTGRAY + (BLACK << 4));
    changecursor(insert);
    do {
        gotoxy(posx, posy);
        for( i = 0; i < maxlength; i++) /* clear to the end of line */
            putchar(' ');
        gotoxy(posx, posy);
        cputs(s);
        gotoxy((pos+posx), posy);
        switch(c = system_manager(EDITING))
        {
            case Home: pos = 0;
                        break;

            case EndKey: pos = len;
                        break;

            case Ins: insert = !insert;
                     changecursor(insert);
                     break;

            case Larr: if( pos > 0 )
                        pos--;
                        break;

            case Rarr: if( pos < len )
                        pos++;
                        break;

            case BackSp: if( pos > 0 )
                        {
                            movmem(&s[pos], &s[pos-1], len-pos+1);
                            pos--;
                            len--;
                        }
                        break;

            case Del: if( pos < len )
                        {
                            movmem(&s[pos+1], &s[pos], len-pos);
                            len--;
                        }
                        break;

            case Enter: break;

            case Uarr: break;

            case Darr: break;

            case Esc: len = 0;
                     break;

            default: ch = c;
                     if( ((legal[0] == 0) || ( strchr(legal, ch) != NULL)) &&

```

```

        ((ch >= ' ') && (ch <= '-')) &&
        (len < maxlength) ||
        ((len == maxlength) && (!insert)) ,
    {
        if( insert )
        {
            memmove(&s[pos+1], &s[pos], len - pos+1);
            len++;
            s[pos++] = ch;
        }
        else /* overwrite */
        {
            if( (pos >= len) && (len < maxlength) )
                len++;
            s[pos++] = ch;
            if( pos >= maxlength ) pos--;
        }
    }
    break;
} /* end switch */
} while( (c != Enter) && (c != Esc) );
s[len] = 0;
hidecursor(); /* remove cursor */
return(r    Esc);
}

/*****. *****/
/*edit_message: let the user edit 4 lines by 33 character message */
int edit_message(void)
{
    int c, i, len[4], pos, insert = 1, posx = 2, posy = 3, maxlength = 33;
    int line = 0;
    char ch;

    for( i = 0; i < 3; i++)
        len[i] = strlen(mess_text[i]);
    pos = len[0];
    textattr(WHITE + (CYAN << 4));
    changecursor(insert);
    window(1,19,36,25);
    do {
        gotoxy(posx, posy);
        for( i = 0; i < maxlength; i++) /* clear to the end of line */
            putchar(' ');
        gotoxy(posx, posy);
        puts(mess_text[line]);
        gotoxy((pos+posx), posy);
        switch(c = system_manager(EDITING))
        {
            case Home: pos = 0;
                        break;

            case EndKey: pos = len[line];
                        break;

            case Ins: insert = !insert;
                     changecursor(insert);
                     break;

            case Larr: if( pos > 0 )
                        pos--;
                        break;

            case Rarr: if( pos < len[line] )
                        pos++;
                        break;

            case Uarr: if( line > 0 )
                        {
                            line--;
                            posy--;
                            if(len[line] < pos)
                                pos = len[line];
                        }
                        break;

            case Darr: if( line < 2 )
                        {
                            line++;
                            posy++;
                            if(len[line] < pos)
                                pos = len[line];
                        }
        }
    } while( c != Esc );
}

```

```

        break;
    case BackSp: if( pos > 0 )
        {
            movmem(&mess_text[line][pos], &mess_text[line][pos-1],
                    len[line]-pos+1);
            pos--;
            len[line]--;
        }
        break;
    case Del: if( pos < len[line] )
        {
            movmem(&mess_text[line][pos+1], &mess_text[line][pos],
                    len[line]-pos);
            len[line]--;
        }
        break;
    case Enter: if( line < 2 )
        {
            line++;
            posy++;
            if(len[line] < pos)
                pos = len[line];
        }
        break;
    case Esc: len[line] = 0;
        break;
    case CtrlF: break;
    default: ch = c;
        if( ((legal[0] == 0) || (strchr(legal,ch) != NULL)) &&
            ((ch >= ' ' && (ch <= '~')) &&
             (len[line] < maxlength) ||
             ((len[line] == maxlength) && (!insert)) )
        {
            if( insert )
            {
                memmove(&mess_text[line][pos+1], &mess_text[line][pos],
                        len[line] - pos+1);
                len[line]++;
                mess_text[line][pos++] = ch;
            }
            else
            {
                if( (pos >= len[line]) && (len[line] < maxlength) )
                    len[line]++;
                mess_text[line][pos++] = ch;
                if( pos >= maxlength ) pos--;
            }
        }
        break;
    } /* end switch */
} while( (c != CtrlE) && (c != Esc) );
mess_text[line][len[line]] = 0;
hidecursor(); /* remove cursor */
return(c != Esc);
}

```

```

/*-----*/
/* String execution manager code */
/*-----*/

```

```

void string_em (int service, char s[], char *para)
{
    int x, y, maxl;
    char *temp;

    switch (service) {
        case INIT: initcursor();
            break;

        case EDIT: temp = strtok(para, SEPARATOR1);
            if( temp == NULL ) error_em(STRING_MOD);
            x = atoi(temp);
            temp = strtok(NULL, SEPARATOR1);
            if( temp == NULL ) error_em(STRING_MOD);
            y = atoi(temp);
            temp = strtok(NULL, SEPARATOR1);

```



```
        if( temp == NULL ) error_em(STRING_MOD);
        maxl = atoi(temp);
        editstring(s,x,y,maxl);
        break;

    case HIDE: hidecursor();
        break;

    case EDIT_MESS: edit_message( );
        break;

} /* end switch */
```

```

                *****
                * ERROR.EM *
                *****

/* Error Execution Module: The error EM is mainly used for debugging *
/* it reports which module has a problem. It is also used to        *
/* report some warning to the user (i.e. Appointment already exist). *

#include <conio.h>

                *****
                /* error reported */
                /*-----*/

#define INIT_MOD      0
#define TIME_MOD      1
#define WINDOW_MOD    2
#define AGENDA_MOD    3
#define STRING_MOD    4
#define NO_MOUSE      5
#define NO_MATCH      6
#define TIME_CONFLICT 7
#define INCOMPLETE    8

                /*-----*/
                /* module constants */
                /*-----*/

#define LTCORNER      0xc9
#define RTCORNER      0xbb
#define LBCORNER      0x9
#define RBCORNER      0xc
#define HORIZONTAL     0xcd
#define VERTICAL       0xba

/*****/
void error_em ( int error_no )
{
    char buf[4*35*2];
    int i;

    gettext(25,12,55,15,buf);
    textattr(WHITE + (RED<<4));
    window(25,12,55,14);
    clrscr();
    window(25,12,55,15);
    putch(LTCORNER);
    for( i = 26; i < 55; i++)
        putch(HORIZONTAL);
    putch(RTCORNER);
    gotoxy(1,2);
    putch(VERTICAL);
    gotoxy(31,2);
    putch(VERTICAL);
    gotoxy(1,3);
    putch(LBCORNER);
    for( i = 26; i < 55; i++)
        putch(HORIZONTAL);
    putch(RBCORNER);
    sound(800);
    gotoxy(3,2);
    switch (error_no) {

        case INIT_MOD: cputs("ERROR: Initialize Module");
                        break;
        case TIME_MOD: cputs("ERROR: Time Module");
                        break;
        case WINDOW_MOD: cputs("ERROR: Window Module");
                        break;
        case AGENDA_MOD: cputs("ERROR: Agenda Module");
                        break;
        case STRING_MOD: cputs("ERROR: String Module");
                        break;
        case NO_MOUSE: cputs("NO MOUSE DETECTED");
                        break;
        case NO_MATCH: cputs("Appointment not found");
                        break;
        case TIME_CONFLICT: cputs("Appt. Time Conflict");
                        break;
        case INCOMPLETE: cputs("Incomplete appt. Data");
                        break;
    }
}

```

APPENDIX A
Personal Agenda code listing
184

```
delay(400);  
nosound();  
delay(2000);  
puttext(25, 12, 55, 15, buf);  
}
```

```

/******
/* file: AGENDA.EM */
/******

/* Agenda execution module: The Agenda EM contains the various
/* file services (like OPEN, CLOSE), and the various operations
/* on the Agenda structures which are defined in the file
/* "AGENDA.DAT".

#include <STDIO.H>
#include <STRING.H>

/******
/* Agenda EM services */
/*-----*/

#define OPEN      0 /* open agenda file AG##.$$$ */
#define CLOSE    1 /* close agenda file */
#define SAVE      2 /* save Agenda structure */
#define RETRIEVE  3 /* retrieve Agenda structure */
#define INSERT    4 /* insert a new appointment */
#define ADELETE   5 /* delete an appointment */
#define SETVAR    6 /* set the current variables */
#define MATCH_ST  7 /* match start time with an appt in the current day */
#define MATCH_ET  8 /* as MATCH_ST but with end_time */

/******
/* Agenda EM constant declaration */
/*-----*/

#define SUCCESS 1
#define NOTFOUND 0
#define ASEPARATOR " ./"
#define FS 0x1c /* field separator characters */
#define legal " ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz\
0123456789_$&.,:-!*@!~?+/( ) { } [ ] % < > ="

/******
/* Agenda EM procedures/functions definition */
/*-----*/

/******
/* new_year: Allocates memory for a new year structure and returns
/* a pointer to this new structure.
*/

PYears new_year (void)
{
    return( (PYears)(malloc(sizeof(struct years))) );
}

/******
/* new_month: Allocates memory for a new month structure and returns
/* a pointer to this new structure.
*/

PMonths new_month (void)
{
    return( (PMonths)(malloc(sizeof(struct months))) );
}

/******
/* new_day: Allocates memory for a new day structure and returns
/* a pointer to this new structure.
*/

PDays new_day (void)
{
    return( (PDays)(malloc(sizeof(struct days))) );
}

/******
/* new_appointment: Allocates memory for a new appointment structure
/* and returns a pointer to this new structure.
*/

PAppt new_appointment (void)
{
    return( (PAppt)(malloc(sizeof(struct appt))) );
}

/******
/* save_agenda: saves the complete agenda structure into file
/* AG##.$$$
*/

void save_agenda (void)

```

```

{
    Pyears iy;
    PMonths im;
    PDays id;
    PAppt ia;
    int i, j;

    flushall();
    for( i = 0; i < strlen(ux); i++ )
    {
        j = ux[i] + 128;
        flushall();
        fprintf(ag, "%d ", j);
    }
    flushall();
    fprintf(ag, "%d ", -1);
    for( i = 0; i < strlen(sx); i++ )
    {
        j = sx[i] + 128;
        flushall();
        fprintf(ag, "%d ", j);
    }
    flushall();
    fprintf(ag, "%d\n", -1);
    i = 0;
    while( i < 3 )
    {
        if( strlen(mess_text[i]) != 0 )
        {
            flushall();
            fprintf(ag, "%s\n", mess_text[i++]);
        }
        else
            i++;
    }
    flushall();
    fprintf(ag, "%c\n", FS);
    iy = /agenda;
    while( iy != NULL )          /* save all years in the agenda */
    {
        flushall();
        fprintf(ag, "%s\n", iy->yyyy);          /* save year value */
        im = iy->month set;
        while( im != NULL )      /* save all month for that year */
        {
            flushall();
            fprintf(ag, "%s\n", im->mm);          /* save month number */
            id = im->day set;
            while( id != NULL )  /* save all days for this month */
            {
                flushall();
                fprintf(ag, "%s\n", id->id);          /* save day date */
                ia = id->appointment;
                while( ia != NULL ) /* save all appointment for this day */
                {
                    flushall();
                    fprintf(ag, "%s\n%s\n%s\n%d\n", ia->start_time, ia->end_time,
                        ia->comment, ia->private);
                    ia = ia->next;
                }
                fprintf(ag, "%c\n", FS); /* mark end of appointment saved */
                id = id->next;
            }
            fprintf(ag, "%c\n", FS); /* mark end of days saved */
            im = im->next;
        }
        fprintf(ag, "%c\n", FS); /* mark end of months saved */
        iy = iy->next;
    }
}

/*****
/* retrieve agenda: retrieves the complete agenda structure from
/* file AG##.$$$
*/
*/

void retrieve_agenda (void)
{
    PYears iy, priy;
    PMonths im, prim;
    PDays id, prid;
    PAppt ia, pria;
    char temp[37];
    int i, j;

```

```

strcpy(ux,"");          /* clear the passwords first */
strcpy(sx,"");
i = 0;
while( i < 3 )
    strcpy(messe_text[i++],"");
i = 0;
fscanf(ag,"%d",&j);
if( ((j-128) > '~') && ((j-128) < '~') )
{
    while( j != -1 )
    {
        ux[i++] = j - 128;
        fscanf(ag,"%d",&j);
    }
    i = 0;
    fscanf(ag,"%d",&j);
    while( j != -1 )
    {
        sx[i++] = j - 128;
        fscanf(ag,"%d",&j);
    }
    i = 0;
    while( fgets(temp,37,ag) )
    {
        if( strchr(temp,FS) != NULL )
            break;
        if( strcmp(temp,"\n") != 0 )
            strcpy(messe_text[i++],temp);
    }
}
else /* no password in the file ag##.$$$ */
{
    /* reset the file for reading other data in proper order */
    close(ag);
    ag = fopen("AG##. $$$", "rt");
}
iy = new_year();
Agenda = iy;
while( fscanf(ag,"%s",iy->iyyy) != EOF ) /* for all the years saved */
{
    im = new_month();
    iy->month_set = im;
    do {
        fscanf(ag,"%s",im->mm) /* retrieve all the months */
        if( strchr(im->mm,FS) != NULL )
        {
            /* when the end marker is reached, find the previous */
            /* month in the structure to mark the end of the */
            /* list with a NULL */
            prim = iy->month_set;
            while( prim->next != im ) prim = prim->next;
            prim->next = NULL;
            im = NULL;
            break;
        }
    }
    id = new_day();
    im->day_set = id;
    do {
        fscanf(ag,"%s",id->id) /* retrieve all the days */
        if( strchr(id->id,FS) != NULL )
        {
            /* when the end marker is reached, find the previous */
            /* day in the structure to mark the end of the list */
            /* with a NULL */
            prid = im->day_set;
            while( prid->next != id ) prid = prid->next;
            prid->next = NULL;
            id = NULL;
            break;
        }
    }
    ia = new_appointment();
    id->appointment = ia;
    strcpy(temp,"0"); /* make sure to start with a scan of file */
    do {
        if( (strcmp(temp,"0") == 0) || (strcmp(temp,"1") == 0) )
            fscanf(ag,"%s",ia->start_time);
        else
            strcpy(ia->start_time,temp);
        if( strchr(ia->start_time,FS) != NULL )
        {
            /* when the end marker is reached, find the previous */
            /* appointment in the struct. to mark the end of the */
            /* list with a NULL */
            pria = id->appointment;
            while( pria->next != ia ) pria = pria->next;
        }
    }
}

```

```

        pria->next = NULL;
        ia = NULL;
        break; /* break moves you out of the loop */
    }
    fscanf(ag, "%s\n", ia->end_time);
    fgets(ia->comment, 36, ag);
    fscanf(ag, "%s\n", temp);
    if( (strcmp(temp, "0") == 0) || (strcmp(temp, "1") == 0) )
        ia->private = atoi(temp); /* check for upward compability */
    else
        ia->private = 0;
    ia->next = new_appointment();
    ia = ia->next;
    } while( 1 );
    id->next = new_day();
    id = id->next;
    } while( 1 );
    im->next = new_month();
    im = im->next;
    } while( 1 );
    iy->next = new_year();
    iy = iy->next;
}
priy = Agenda;
if( priy == iy )
    Agenda = NULL; /* file ag##.$$$ was empty */
else
{
    while( priy->next != iy ) priy = priy->next;
    priy->next = NULL;
}
iy = NULL;
}

/*.....*/
/* set current variable: Sets the cyear, cmonth and cday variables */
/* according to the current_year, current_month and current_day strings */

int set_current_variable (void)
{
    int flag1 = 0, flag2 = 0;

    if( Agenda == NULL ) /* empty structure */
        return(SUCCESS);
    if( strcmp(cyear->yyyy, current_year) != 0 )
    {
        cyear = Agenda;
        while( cyear != NULL )
        {
            if( strcmp(cyear->yyyy, current_year) == 0 )
                break;
            cyear = cyear->next;
        }
        flag1 = 1;
    }
    if( flag1 || (strcmp(cmonth->mm, current_month) != 0) )
    {
        cmonth = cyear->month set;
        while( cmonth != NULL )
        {
            if( strcmp(cmonth->mm, current_month) == 0 )
                break;
            cmonth = cmonth->next;
        }
        flag2 = 1;
    }
    if( flag2 || (strcmp(cday->id, current_day) != 0) )
    {
        cday = cmonth->day set;
        while( cday != NULL )
        {
            if( strcmp(cday->id, current_day) == 0 )
                break;
            cday = cday->next;
        }
    }
    return(SUCCESS);
}

/*.....*/
/* insert_appointments: Inserts a new appointment in the Agenda */
/* structure. It takes into account all possibilities in the data */
/* structure. */

```

```

int insert_appointment (char *parameters)
{
    char *st, *et, *comment, *priv;

    /* Break down the parameter string */
    st = strtok(parameters, ASEPARATOR);
    if( st == NULL ) return(NOEVENT);
    et = strtok(NULL, ASEPARATOR);
    if( et == NULL ) return(NOEVENT);
    priv = strtok(NULL, ASEPARATOR);
    comment = strtok(NULL, ASEPARATOR);
    if( Agenda == NULL ) /* then start a new agenda */
    {
        Agenda = new year();
        strcpy(Agenda->yyy, current_year);
        Agenda->next = NULL;
        cyear = Agenda; /* set the current year pointer */
        Agenda->month_set = new month();
        cmonth = Agenda->month_set;
        strcpy(cmonth->mm, current_month);
        cmonth->next = NULL;
        cmonth->day_set = new day();
        cday = cmonth->day_set;
        strcpy(cday->id, current_day);
        cday->next = NULL;
        cday->appointment = new appointment();
        cappointment = cday->appointment;
        strcpy(cappointment->start_time, st);
        strcpy(cappointment->end_time, et);
        strcpy(cappointment->comment, comment);
        cappointment->private = atoi(priv);
        cappointment->next = NULL;
        return(SUCCESS);
    }
    /* else find if year, month, and date already exist */
    set_current_variable();
    if( cyear == NULL ) /* year is not in the agenda */
    {
        cyear = Agenda;
        while( cyear->next != NULL ) /* find the last year record */
            cyear = cyear->next;
        cyear->next = new year(); /* add a record to the end */
        cyear = cyear->next; /* go to new record */
        strcpy(cyear->yyy, current_year);
        cyear->next = NULL;
        cyear->month_set = new month(); /* add a new month record */
        cmonth = cyear->month_set; /* go to new record */
        strcpy(cmonth->mm, current_month);
        cmonth->next = NULL;
        cmonth->day_set = new day(); /* add a new day record */
        cday = cmonth->day_set; /* go to new record */
        strcpy(cday->id, current_day);
        cday->next = NULL;
        cday->appointment = new appointment(); /* make a new appointment */
        cappointment = cday->appointment;
        strcpy(cappointment->start_time, st);
        strcpy(cappointment->end_time, et);
        strcpy(cappointment->comment, comment);
        cappointment->private = atoi(priv);
        cappointment->next = NULL;
        return(SUCCESS);
    }
    if( cmonth == NULL ) /* year exist but not month */
    {
        cmonth = cyear->month_set;
        while( cmonth->next != NULL ) /* find the last year record */
            cmonth = cmonth->next;
        cmonth->next = new month(); /* add a new record to it */
        cmonth = cmonth->next; /* go to new record */
        strcpy(cmonth->mm, current_month);
        cmonth->next = NULL;
        cmonth->day_set = new day(); /* add a new day record */
        cday = cmonth->day_set; /* go to new record */
        strcpy(cday->id, current_day);
        cday->next = NULL;
        cday->appointment = new appointment(); /* make a new appointment */
        cappointment = cday->appointment;
        strcpy(cappointment->start_time, st);
        strcpy(cappointment->end_time, et);
        strcpy(cappointment->comment, comment);
        cappointment->private = atoi(priv);
        cappointment->next = NULL;
        return(SUCCESS);
    }
}

```



```

if( cday == NULL ) /* year and month exist but not day */
{
    cday = cmonth->day_set;
    while( cday->next != NULL ) /* find the last year record */
        cday = cday->next;
    cday->next = new day(); /* add a new record to it */
    cday = cday->next; /* go to new record */
    strcpy(cday->id, current_day);
    cday->next = NULL;
    cday->appointment = new appointment(); /* make a new appointment */
    cappointment = cday->appointment;
    strcpy(cappointment->start_time, st);
    strcpy(cappointment->end_time, et);
    strcpy(cappointment->comment, comment);
    cappointment->private = atoi(priv);
    cappointment->next = NULL;
    return(SUCCESS);
}
cappointment = cday->appointment; /* else check for time conflict */
while( cappointment != NULL )
{
    if( ((strcmp(cappointment->start_time, st) <= 0) &&
        (strcmp(cappointment->end_time, et) >= 0)) ||
        ((strcmp(cappointment->start_time, et) <= 0) &&
        (strcmp(cappointment->end_time, st) >= 0)))
        return(TIME_BOOKED);
    if( cappointment->next != NULL )
        cappointment = cappointment->next;
    else
        break;
}
cappointment->next = new appointment(); /* make a new appointment */
cappointment = cappointment->next;
strcpy(cappointment->start_time, st);
strcpy(cappointment->end_time, et);
strcpy(cappointment->comment, comment);
cappointment->private = atoi(priv);
cappointment->next = NULL;
return(SUCCESS);
}

/*****
/* delete appointment: Deletes an appointment from the agenda structure*/
/* Appointments can only be deleted in the current_year, current_month */
/* and current_day.
int delete_appointment (char *parameters)
{
    PYears py; /* p stands for the previous */
    PMonths pm;
    PDays pd;
    PAppt ia, pa;

    if( Agenda == NULL ) return(SUCCESS); /* agenda is already empty */
    set_current_variable();
    if( (cyear == NULL) || (cmonth == NULL) || (cday == NULL) )
        return(NOTFOUND);
    ia = cday->appointment;
    while( ia != NULL )
        if( strcmp(ia->start_time, parameters) == 0 )
            break;
        else
            ia = ia->next;
    if( ia == NULL ) return(NOTFOUND);
    /*** at this point the appointment has been located ***/
    pa = cday->appointment;
    if( pa == ia )
        cday->appointment = ia->next;
    else
    {
        while( pa->next != ia ) pa = pa->next;
        pa->next = ia->next;
    }
    if( cday->appointment == NULL )
    {
        pd = cmonth->day_set;
        if( pd == cday )
            cmonth->day_set = cday->next;
        else
        {
            while( pd->next != cday ) pd = pd->next;
            pd->next = cday->next;
        }
    }
}

```

```

    else
        return(SUCCESS);
    if( cmonth->day_set == NULL )
    {
        pm = cyear->month_set;
        if( pm == cmonth )
            cyear->month_set = cmonth->next;
        else
        {
            while( pm->next != cmonth ) pm = pm->next;
            pm->next = cmonth->next;
        }
    }
    else
        return(SUCCESS);
    if( cyear->month_set == NULL )
    {
        py = Agenda;
        if( py == cyear )
            Agenda = cyear->next;
        else
        {
            while( py->next != cyear ) py = py->next;
            py->next = cyear->next;
        }
    }
    return(SUCCESS);
}

/*****
/* match appointment: Looks for an appointment in the structure, the
/* search key is either start_time or end_time.
*****/

int match_appointment( int match_select, char *param )
{
    cappointment = NULL;          /* make sure make new search */
    if( Agenda == NULL )
        return(NOTFOUND);
    cyear = Agenda;
    while( strcmp(cyear->yyyy, current_year) != 0 )
    {
        cyear = cyear->next;
        if( cyear == NULL )
            return(NOTFOUND);
    }
    cmonth = cyear->month_set;
    while( strcmp(cmonth->mm, current_month) != 0 )
    {
        cmonth = cmonth->next;
        if( cmonth == NULL )
            return(NOTFOUND);
    }
    cday = cmonth->day_set;
    while( strcmp(cday->id, current_day) != 0 )
    {
        cday = cday->next;
        if( cday == NULL )
            return(NOTFOUND);
    }
    cappointment = cday->appointment;
    switch( match_select ) {
        case MATCH_ST: while( strcmp(cappointment->start_time, param) != 0 )
        {
            cappointment = cappointment->next;
            if( cappointment == NULL )
                return(NOTFOUND);
        }
        break;
        case MATCH_ET: while( strcmp(cappointment->end_time, param) != 0 )
        {
            cappointment = cappointment->next;
            if( cappointment == NULL )
                return(NOTFOUND);
        }
        break;
    }
    return(SUCCESS);
}

/*****
/* Agenda execution manager code */
*****/

```

```

/*-----*/
int agenda_em ( int service, char *parameters )
{
    char *name, *mode;
    switch ( service ) {
        case OPEN: name = strtok(parameters, ASEPARATOR);
                    if( name == NULL ) return(0);
                    mode = strtok(NULL, ASEPARATOR);
                    if( mode == NULL ) mode = "rt";
                    if( (ag = fopen(name, mode)) == NULL )
                    {
                        ag = fopen(name, "wt");
                        ag = freopen(name, "rt", ag);
                    }
                    return(SUCCESS); /* file successfully opened */
        case CLOSE: fclose(ag);
                    return(SUCCESS);
        case SAVE: name = strtok(parameters, ASEPARATOR);
                    if( name == NULL ) return(0);
                    ag = freopen(name, "wt", ag);
                    save_agenda();
                    return(SUCCESS);
        case RETRIEVE: retrieve_agenda();
                       return(SUCCESS);
        case INSERT: return(insert_appointment(parameters));
        case ADELETE: return(delete_appointment(parameters));
        case SETVAR: return(set_current_variable());
        case MATCH_ST:
        case MATCH_ET: return(match_appointment( service, parameters));
    } /* end switch( service ) */
}

```

```

/* *****
/* file: WINDOW.EM */
/* *****

/* Window execution manager: The window EM contains all pertinent data */
/* associated with windows in the agebda system. This includes: what */
/* the windows look like; where the significant coordinate are for the */
/* mouse button; where should data be written when received; etc... */
/* The file "windobj.c" contains all the generic procedures/functions */
/* used by the window EM. */

#include "windlib.c" /* window library of procedures/functions */

/* *****
/* Window EM services */
/* *****

#define NOSERVICE -1 /* return value of no service executed */
#define INITIALIZE 0 /* initialize system windows */
#define FILLMONTH 1 /* fill month window data */
#define FILLDAY 2 /* fill day window data */
#define OPEN_ADD 3 /* open ADD function window */
#define OPEN_DELETE 4 /* open DELETE function window */
#define OPEN_MODIFY 5 /* open MODIFY function window */
#define COORD_TO_EVENT 6 /* translate mouse coord. to event */
#define CLOSE_WINDOW 7 /* close active window */
#define HIGHLIGHT 8 /* highlight the current text box */
#define SETAPP 9 /* highlight days with appt in month window */
#define FILL_COM 10 /* fill the function window with current */
/* appointment data */
#define OPEN_PASS 11 /* open the window for reading the password */
#define OPEN_NEW_PASS 12 /* open window for a changing the password */
#define OPEN_NOPASS 13 /* open window with no password message */
#define WRITELN 14 /* change message in window */
#define OPEN_MESS 15 /* open the message window */
#define TWO_MONTH 16 /* initialize the two month windows */
#define FLASH 17 /* make current day flash or stop flashing */

/* *****
/* Window EM constant declaration */
/* *****

#define ON 1
#define OFF 0

/* *****
/* Window EM global variables */
/* *****

int current_box;
int calendar_square;

/* *****
/* Window EM procedures and functions */
/* *****
/* command_line: Puts the two command <CANCEL> <OK> at the coordinate */
/* x and y. */

void command_line( int x, int y )
{
    gotoxy(x,y);
    cputs("<CANCEL> <OK>");
}

/* *****
/* open_message_window: opens the message window an copies the message */
/* in it. */

void open_message_window (void)
{
    open_window(1,19,36,25,CYAN,1,"MESSAGE");
    window(1,19,36,25);
    gotoxy(2,3);
    cputs(mess_text[0]);
    gotoxy(2,4);
    cputs(mess_text[1]);
    gotoxy(2,5);
    cputs(mess_text[2]);
}

/* *****
/* message_print: prints different messages in the different windows */
/* *****

```

```

/* it is used to change specific message. */
void message_print(char *param)
{
    int message_number;

    message_number = atoi(param);
    switch(message_number)
    {
        case 1: textattr(WHITE + (RED << 4) );
                gotoxy(3,2);
                cputs(" Enter Secretary Password ");
                break;
        case 2: textattr(WHITE + (RED << 4) );
                gotoxy(3,2);
                cputs(" Enter New User Password ");
                break;
        case 3: textattr(WHITE + (RED << 4) );
                gotoxy(3,2);
                cputs(" New Secretary Password ");
                break;
        default: break;
    }
}

/*****
/* init agenda window: initialize the two main agenda windows, calendar */
/* and daily schedule window. */
void init_agenda_windows(void)
{
    clrscr();
    puttext(4,2,32,18,CALGRID);
    puttext(37,1,76,20,DAYGRID1);
    puttext(37,21,76,25,DAYGRID2);
    window(37,1,76,25);
    gotoxy(12,23);
    textattr(WHITE + (BLUE << 4) );
    cputs("<PLAN> ");
    if( level == 1 )
        cputs("<PASSWORD> <MESSAGE>");
    else
    {
        textattr(LIGHTGRAY + (BLUE << 4) );
        cputs("<PASSWORD>");
        gotoxy(15,24);
        cputs("<ADD> <DEL> <MOD>");
        gotoxy(30,23);
        textattr(WHITE + (BLUE << 4) );
        cputs(" <MESSAGE>");
    }
}

/*****
/* open function window: open the window for the three function in the */
/* agenda system: ADD, DELETE, MODIFY. The windows are similar, only */
/* the title is different. */
void open_function_window(int typ)
{
    char *title;

    if( window_number == 10 ) return;
    7*( x1, y1, x2, y2,FRAME, Background, Title )*/
    switch ( typ ) {
        case OPEN_ADD: title = "Add Appointment";
                        break;
        case OPEN_DELETE: title = "Delete Appointment";
                        break;
        case OPEN_MODIFY: title = "Modify Appointment";
                        break;
        default: title = "";
                break;
    }
    open_window(5,6,75,15,BLUE,1,title);
    window(5,6,75,15);
    textattr(LIGHTGRAY + (BLUE<<4));
    text_box(8,5,5,"Start"); /*( x, y, length, Title)*/
    textattr(LIGHTGRAY + (BLUE<<4));
    text_box(17,5,5,"End");
    textattr(LIGHTGRAY + (BLUE<<4));
    text_box(25,5,36,"Comment");
    current_box = 1;
    highlight_box(current_box, ON);
}

```

APPENDIX A

Personal Agenda code listing

195

```

textattr(WHITE + (BLUE <<4));
command_line(45,8); /* writes "<Cancel> .OK>" at ( x, y ) *
if( typ == OPEN_ADD )
{
    gotoxy(35,8);
    cputs("<PRIVATE>");
}
}

/*****
/* open_password_window: open the window for the reading or changing */
/* system password */
void open_password_window (int typ)
{
    char *title;

    switch ( typ ) {
        case OPEN_PASS:    title = "Please Enter Password";
                           break;
        case OPEN_NEW_PASS: title = "Enter Old User Password";
                           break;
        case OPEN_NOPASS:  title = "Please Set User Password";
                           break;
    }
    open_window(24,10,56,16,RED,1,title);
    window(24,10,56,16);
    textattr(WHITE + (RED <<4));
    text_box(12,4,9,""); /* ( x, y, length, Title)*/
    textattr(WHITE + (BLUE <<4));
}

/*****
/* init_two_month_windows: Initializes the two month windows for the */
/* PLANNING mode operation. */
void init_two_month_windows(void)
{
    window(1,1,80,25);
    textattr(WHITE + (BLACK << 4));
    clrscr();
    textattr(WHITE + (BLUE << 4));
    puttext(4,2,32,18,CALGRID);
    puttext(38,2,66,18,CALGRID);
    window(38,2,66,18);
    gotoxy(12,16);
    cputs(" ");
    mode = PLANNING;
}

/*****
/* coordinate_to_event: Returns the event that corresponds to the mouse */
/* button coordinates passed in by parameter. */
int coordinate_to_event( char *parameter )
{
    int x, y, t;
    char *temp;

    temp = strtok(parameter," ");
    if( temp == NULL ) return(NOEVENT);
    x = atoi(temp);
    temp = strtok(NULL," ");
    if( temp == NULL ) return(NOEVENT);
    y = atoi(temp);
    switch ( window number ) {
        /* window 1 to 10 = in a function (i.e. Add, Delete etc.) */
        case 1:
        case 2:
        case 3:
        case 4:
        case 5:
        case 6:
        case 7:
        case 8:
        case 9:
        case 10: switch ( y ) {
            case 9: if( (x >= 11) && (x <= 15) ) /* start time text box */
                    return(START_TIME);
                    if( (x >= 21) && (x <= 25) ) /* end time text box */
                    return(END_TIME);
                    if( (x >= 28) && (x <= 63) ) /* comment text box */
                    return(COMMENT);
                    return(NOEVENT); /* x not in range */

```

```

        case 12: if( (x >= 49) && (x <= 56) )
                  return(CANCEL); /* cancel operation */
                  if( (x >= 58) && (x <= 61) )
                    return(OK); /* terminate normally */
                  if( (x >= 38) && (x <= 47) )
                    return(PRIVATE);
                  return(NOEVENT); /* x not in range */
        } /* end switch( y ) */
default:
  if( (mode == PLANNING) && ((x >= 38) && (x <= 56)) )
  {
    x -= 34;
    into = 2;
  }
  else
    select = 1;
  switch( y ) {

    case 3: /* Scroll up arrow for appointment window */
      if( mode == NORMAL )
      {
        if( ( x == 70 ) ||
            ( x == 71 ) ||
            ( x == 72 ) )
          return(SCROLL_UP);
        else break;
      }
      break;
    case 6: /* Calendar line 1 */
      switch( x ) {
        case 4: /* Sunday coordinates */
        case 5:
          calendar_square = 1;
          return(SELECT_DAY);

        case 8: /* Monday coordinates */
        case 9:
          calendar_square = 2;
          return(SELECT_DAY);

        case 12: /* Tuesday coordinates */
        case 13:
          calendar_square = 3;
          return(SELECT_DAY);

        case 16: /* Wednesday coordinates */
        case 17:
          calendar_square = 4;
          return(SELECT_DAY);

        case 20: /* Thursday coordinates */
        case 21:
          calendar_square = 5;
          return(SELECT_DAY);

        case 24: /* Friday coordinates */
        case 25:
          calendar_square = 6;
          return(SELECT_DAY);

        case 28: /* Saturday coordinates */
        case 29:
          calendar_square = 7;
          return(SELECT_DAY);
        case 30:
          calendar_square = 7;
          return(SELECT_DAY);
      } /* end switch ( x ) */
      return(NOEVENT); /* calendar line 1 out of x range */
    case 8: /* Calendar line 2 */
      switch( x ) {
        case 4: /* Sunday coordinates */
        case 5:
          calendar_square = 8;
          return(SELECT_DAY);

        case 8: /* Monday coordinates */
        case 9:
          calendar_square = 9;
          return(SELECT_DAY);

        case 12: /* Tuesday coordinates */
        case 13:
          calendar_square = 10;
          return(SELECT_DAY);
      }

```

APPENDIX A
Personal Agenda code listing
197

```
case 16: /* Wednesday coordinates */
case 17:
case 18: calendar square = 11;
        return(SELECT_DAY);

case 20: /* Thursday coordinates */
case 21:
case 22: calendar square = 12;
        return(SELECT_DAY);

case 24: /* Friday coordinates */
case 25:
case 26: calendar square = 13;
        return(SELECT_DAY);

case 28: /* Saturday coordinates */
case 29:
case 30: calendar square = 14;
        return(SELECT_DAY);
} /* end switch ( x ) */
return(NOEVENT); /* calendar line 2 out of x range */

case 10: /* Calendar line 3 */
switch ( x ) {
case 4: /* Sunday coordinates */
case 5:
case 6: calendar square = 15;
        return(SELECT_DAY);

case 8: /* Monday coordinates */
case 9:
case 10: calendar square = 16;
        return(SELECT_DAY);

case 12: /* Tuesday coordinates */
case 13:
case 14: calendar square = 17;
        return(SELECT_DAY);

case 16: /* Wednesday coordinates */
case 17:
case 18: calendar square = 18;
        return(SELECT_DAY);

case 20: /* Thursday coordinates */
case 21:
case 22: calendar square = 19;
        return(SELECT_DAY);

case 24: /* Friday coordinates */
case 25:
case 26: calendar square = 20;
        return(SELECT_DAY);

case 28: /* Saturday coordinates */
case 29:
case 30: calendar square = 21;
        return(SELECT_DAY);
} /* end switch ( x ) */
return(NOEVENT); /* calendar line 3 out of x range */

case 12: /* Calendar line 4 */
switch ( x ) {
case 4: /* Sunday coordinates */
case 5:
case 6: calendar square = 22;
        return(SELECT_DAY);

case 8: /* Monday coordinates */
case 9:
case 10: calendar square = 23;
        return(SELECT_DAY);

case 12: /* Tuesday coordinates */
case 13:
case 14: calendar square = 24;
        return(SELECT_DAY);

case 16: /* Wednesday coordinates */
case 17:
case 18: calendar square = 25;
        return(SELECT_DAY);

case 20: /* Thursday coordinates */
```



```

case 21:
case 22: calendar_square = 26;
        return(SELECT_DAY);

case 24: /* Friday coordinates */
case 25:
case 26: calendar_square = 27;
        return(SELECT_DAY);

case 28: /* Saturday coordinates */
case 29:
case 30: calendar_square = 28;
        return(SELECT_DAY);
    } /* end switch ( x ) */
return(NOEVENT); /* calendar line 4 out of x range */

case 14: /* Calendar line 5 */
switch ( x ) {
case 4: /* Sunday coordinates */
case 5:
case 6: calendar_square = 29;
        return(SELECT_DAY);

case 8: /* Monday coordinates */
case 9:
case 10: calendar_square = 30;
        return(SELECT_DAY);

case 12: /* Tuesday coordinates */
case 13:
case 14: calendar_square = 31;
        return(SELECT_DAY);

case 16: /* Wednesday coordinates */
case 17:
case 18: calendar_square = 32;
        return(SELECT_DAY);

case 20: /* Thursday coordinates */
case 21:
case 22: calendar_square = 33;
        return(SELECT_DAY);

case 24: /* Friday coordinates */
case 25:
case 26: calendar_square = 34;
        return(SELECT_DAY);

case 28: /* Saturday coordinates */
case 29:
case 30: calendar_square = 35;
        return(SELECT_DAY);
    } /* end switch ( x ) */
return(NOEVENT); /* calendar line 4 out of x range */

case 16: /* Calendar line 6 */
switch ( x ) {
case 4: /* Sunday coordinates */
case 5:
case 6: calendar_square = 36;
        return(SELECT_DAY);

case 8: /* Monday coordinates */
case 9:
case 10: calendar_square = 37;
        return(SELECT_DAY);

case 15: /* up one month coordinates */
case 16:
case 17:
case 18:
case 19:
case 20: if( (mode == NORMAL) || (select == 1) )
        return(MONTH_UP);
        else
        break;
case 23: /* down one month coordinates */
case 24:
case 25:
case 26:
case 27:
case 28: if( (mode == NORMAL) || (select == 1) )
        return(MONTH_DOWN);
        else

```

```

        break;
    }
    return(NOEVENT); /* calendar line 6 out of x range */

case 19: /* Scroll down arrow for appointment window */
    if( mode == NORMAL )
    {
        if( ( x == 70 ) ||
            ( x == 71 ) ||
            ( x == 72 ) )
            return(SCROLL_DOWN);
        else break;
    }
    else
        break;
case 22: /* command line 1 coordinate */
    if( mode == PLANNING )
        break;
    if( (x >= 37) && (x <= 52) )
        return(PLAN);
    if( (x >= 54) && (x <= 63) && (level == 1) )
        return(PASSWORD);
    if( (x >= 65) && (x <= 73) )
        return(MESSAGE);
case 23: /* command line 2 coordinate */
    if( mode == PLANNING )
        break;
    switch ( x ) {
        case 50: /* Add command coordinates */
        case 51:
        case 52:
        case 53:
        case 54: return(ADD);

        case 56: /* Delete command coordinates */
        case 57:
        case 58:
        case 59:
        case 60: return(DELETE);

        case 62: /* Modify command coordinates */
        case 63:
        case 64:
        case 65:
        case 66: return(MODIFY);

        case 68: /* Quit command coordinates */
        case 69:
        case 70:
        case 71:
        case 72:
        case 73: return(QUIT);
    } /* end switch ( x ) */
    return(NOEVENT); /* case command line coord out of x range */

default: return(NOEVENT);
} /* end switch ( y ) */
} /* end switch( window_number ) */
}

/*****
/* fill_month_window: This procedure fills in the selected month data */
/* That is the date in the appropriate squares. The variables */
/* current_month and month_offset have to be previously set by the */
/* caller. Month_offset gives the offset from sunday to the first day */
/* of the month. */
void fill_month_window (void)
{
    int i, mm, x, y, year, count;
    div_t d;
    char c[3], yea[5];

    window(4,2,32,18);
    textattr( LIGHTGRAY + (BLUE<<4) );
    mm = atoi(current_month); /* put month name at the top */
    gotoxy(7,2);
    switch (mm) {
        case 1: cputs(" January ");
        case 2: cputs(" February ");
        case 3: cputs(" March ");
    }

```

```

case 4: cputs(" April ");
        break;
case 5: cputs(" May ");
        break;
case 6: cputs(" June ");
        break;
case 7: cputs(" July ");
        break;
case 8: cputs(" August ");
        break;
case 9: cputs("September ");
        break;
case 10: cputs(" October ");
        break;
case 11: cputs(" November ");
        break;
case 12: cputs(" December ");
        break;
    }
    cputs(current_year); /* then put the year at the top */
    cputs(" "); /* clear last character if needed */
    year = atoi(current_year);
    count = 31; /* assume all month have 31 days */
    if( mm == 2 ) /* if february change number of days */
    {
        count = 28;
        d = div(year,4); /* check for leap year */
        if( d.rem == 0 )
            count++;
    }
    else
        if( (mm == 4) || (mm == 6) || (mm == 9) || (mm == 11) )
            count = 30; /* if a month of 30 days change number of days */
    month_size = count; /* save current month size */
    x = 2;
    y = 6;
    for( i = 1; i < 38; i++ ) /* clear window */
    {
        gotoxy(x,y);
        cputs(" ");
        x += 4;
        if( x > 29 )
        {
            x = 2;
            y += 2;
        }
    }
    x = 2 + (4 * month_offset);
    y = 6;
    for( i = 1; i < (count+1); i++ )
    {
        gotoxy(x,y);
        itoa(i, c, 10);
        cputs(c); /* print the number on the calendar */
        x+=4;
        if( x > 29 ) /* if end of line */
        {
            x = 2; /* start next line */
            y+= 2;
        }
    }
}
if( mode == PLANNING )
{
    window(38,7,56,18);
    mm++;
    if( mm == 13 )
    {
        mm = 1;
        year++; /* the second month is in the next year */
        itoa(year,yea,10);
    }
    else
        strcpy(yea,current_year);
    gotoxy(7,2);
    switch ( mm ) {
        case 1: cputs(" January ");
                break;
        case 2: cputs(" February ");
                break;
        case 3: cputs(" March ");
                break;
        case 4: cputs(" April ");
                break;
        case 5: cputs(" May ");
    }
}

```

APPENDIX A
Personal Agenda code listing
201

```

        break;
case 6: cputs("    June ");
        break;
case 7: cputs("    July ");
        break;
case 8: cputs("    August ");
        break;
case 9: cputs("September ");
        break;
case 10: cputs(" October ");
        break;
case 11: cputs(" November ");
        break;
case 12: cputs(" December ");
        break;
    }
    cputs(yea); /* then put the year at the top */
    cputs(" "); /* clear last character if needed */
    count = 31; /* assume all month have 31 days */
    if( mm == 2 ) /* if february change number of days */
    {
        count = 28;
        d = div(year,4); /* check for leap year */
        if( d.rem == 0 )
            count++;
    }
    else
        if( (mm == 4) || (mm == 6) || (mm == 9) || (mm == 11) )
            count = 30; /* if a month of 30 days change number of days */
    next_month_size = count; /* save current month size */
    x = 2;
    y = 6;
    for( i = 1; i < 38; i++ ) /* clear window */
    {
        gotoxy(x,y);
        cputs(" ");
        x += 4;
        if( x > 29 )
        {
            x = 2;
            y += 2;
        }
    }
    x = 2 + (4 * next_month_offset);
    y = 6;
    for( i = 1; i < (count+1); i++ )
    {
        gotoxy(x,y);
        itoa(i, c, 10);
        cputs(c); /* print the number on the calendar */
        x+=4;
        if( x > 29 ) /* if end of line */
        {
            x = 2; /* start next line */
            y+= 2;
        }
    }
}
}

/*****
/* fill_comment: This procedure fills the appointments comments that are
/* in view in the daily schedule window. The time slots are divided
/* into 47 integer indexes, with 0 = 00:00, 1 = 00:30, ..., 46 = 23:00,
/* and 47 = 23:30. Only the first slot of an appointment displays the
/* appointment comment, other slot part of an appointment are simply
/* highlighted in lightgray. The variable cday must have been set to
/* the displayed day before the call to fill_comment.
*****/

void fill_comment (char *bottom)
{
    char *temp, time_copy[6], time_stamp[30], ho[3];
    int top_slot, bot_slot, apt_start, apt_end;
    int flag, y;
    div t dt;
    PAppt apt;

    if( (cday == NULL) || (strcmp(cday->id,current_day) != 0) )
        return; /* if no appointment for that day return */
    window(37,1,76,25);
    textattr(WHITE + (LIGHTGRAY << 4));
    apt = cday->appointment;
    strcpy(time_copy,display_time);
    temp = strtok(time_copy,".");

```

```

top_slot = 2 * atoi(temp);          /* set the top and bottom */
temp = strtok(NULL,":");           /* displayed slot numbers */
if( strcmp("30",temp) == 0 )
    top_slot++;
temp = strtok(bottom,":");
bot_slot = 2 * atoi(temp);
temp = strtok(NULL,":");
if( strcmp("30",temp) == 0 )
    bot_slot++;
while( apt != NULL )                /* check all appointments */
{
    strcpy(time_copy,apt->start_time);
    temp = strtok(time_copy,":");   /* set appointment start */
    apt_start = 2 * atoi(temp);      /* and end slot number */
    temp = strtok(NULL,":");
    if( strcmp("30",temp) == 0 )
        apt_start++;
    strcpy(time_copy,apt->end_time);
    temp = strtok(time_copy,":");
    apt_end = 2 * atoi(temp);
    temp = strtok(NULL,":");
    if( strcmp("30",temp) == 0 )
        apt_end++;
    if( (level == 1) || ((apt->private) && (level == 2)) )
        flag = 1;                  /* flag indicate comment to be written */
    else
        flag = 0;
    while( apt_start <= apt_end )
    {
        /* check all reserved space for appointment */
        if( ((apt_start >= top_slot) && (apt_start <= bot_slot)) ||
            ((top_slot > bot_slot) && ((apt_start <= bot_slot) ||
              (apt_start >= top_slot))))
            /* if in view */
        {
            dt = div(apt_start,2);
            itoa(dt.quot,ho,10);
            if( dt.quot < 10 )
            {
                strcpy(time_stamp,"0");
                strcat(time_stamp,ho);
            }
            else
                strcpy(time_stamp,ho);
            strcat(time_stamp,":");
            if( dt.rem > 0 )
                strcat(time_stamp,"30");
            else
                strcat(time_stamp,"00");
            strcat(time_stamp,"");
            if( (top_slot < bot_slot) || (apt_start >= top_slot) )
                y = 4 + ((apt_start - top_slot) * 3);
            else
                y = 4 + (((47 - top_slot) + bot_slot) * 3);
            gotoxy(4,y);
            cputs(time_stamp);
            if( flag )                /* highkigh time slot */
                /* if comment slot write it in */
            {
                strncpy(time_stamp, apt->comment,30);
                if( (temp = strchr(time_stamp,'\n')) != NULL )
                    *temp = ' ';
                while( strlen(time_stamp) < 30 ) /* pad the comment with blanks */
                    strcat(time_stamp," ");
            }
            else                      /* else simply highligh area */
                strcpy(time_stamp,"");
            gotoxy(4, y+1);
            cputs(time_stamp);
            apt_start++;
            flag = 0;
        }
        apt = apt->next;
    }
}

/*****
/* make day flash: make the current day flash or stops it from flashing */
/* depending on the 'on' parameter value (on == 1 is flash on) */
void make_day_flash( int on )
{
    int iy, wa, x, y;
    div_t d;
    char mon[3], yea[5], c[3];
    PYears cy;

```

APPENDIX A

Personal Agenda code listing

203

```

PMonths cm;
PDays sd;

if( flash_day != 0 )
{
    if( (mode == PLANNING) && (into == 2) )
    {
        window(38,2,66,18); /* in the second month window */
        x = flash_day + next_month_offset;
        y = x - 1;
        iy = atoi(current_year);
        mm = atoi(current_month) + 1;
        if( mm == 13 )
        {
            mm = 1;
            iy++;
        }
    }
    else
    {
        window(4,2,32,18); /* in the first month window */
        x = flash_day + month_offset;
        y = x - 1;
        iy = atoi(current_year);
        mm = atoi(current_month);
    }
    itoa(iy, yea, 10); /* copy of year to look into */
    itoa(mm, mon, 10); /* copy of month to look into */
    while( x > 7 ) x -= 7; /* calculate x coordinate */
    x = 2 + ((x-1) * 4);
    d = div(y,7); /* calculate y coordinate */
    y = 6 + (d * 2);
    cy = Agenda;
    while( cy != NULL )
        if( strcmp(cy->iyyy, yea) == 0 )
            break;
    else
        cy = cy->next;
    if( cy != NULL )
        cm = cy->month_set;
    while( cm != NULL )
        if( strcmp(cm->mm, mon) == 0 )
            break;
    else
        cm = cm->next;
    if( cm != NULL ) /* now check for background highlight */
        sd = cm->day_set;
    else
        sd = NULL;
    itoa(flash_day, c, 10);
    while( sd != NULL )
        if( strcmp(sd->id, c) == 0 ) break;
        else sd = sd->next;
    if( cm == 1 )
        if( sd != NULL ) /* a match was found */
            textattr(WHITE + (LIGHTGRAY << 4) + BLINK);
        else /* no match = no appt that day */
            textattr(LIGHTGRAY + (BLUE << 4) + BLINK);
    else
        if( sd != NULL ) /* a match was found */
            textattr(WHITE + (LIGHTGRAY << 4));
        else /* no match = no appt that day */
            textattr(LIGHTGRAY + (BLUE << 4));
    gotoxy(x,y);
    cputs(c);
}

}

/* ===== */
/* fill_day_window: Fills the daily schedule window. This only puts */
/* the day's date, the times in view, in the window and sets the */
/* flashing day in the month window. It also calls the fill comment */
/* above. The variables current month, current day, month offset and */
/* display_time must be set before fill_day_window is called. */
void fill_day_window (void)
{
    int hour, minute, i, id, x, y, iy, mm;
    char ho[3], min[3], time_copy[6], *temp, c[3];
    char mon[3], yea[5];
    div t d;
    PYears cy;
    PMonths cm;
    PDays sd;

```

```

window(37,1,76,25);
textattr( LIGHTGRAY + 'BLUE'<<4) ;;
id = atoi(current_day);
i = id + 'month_offset - 1';
while( i == 7 ) i -= 7;
gotoxy(9,2);
switch ( i ) {
    case 0: cputs("    Sunday ");
            break;
    case 1: cputs("    Monday ");
            break;
    case 2: cputs("    Tuesday ");
            break;
    case 3: cputs("Wednesday ");
            break;
    case 4: cputs(" Thursday ");
            break;
    case 5: cputs("    Friday ");
            break;
    case 6: cputs(" Saturday ");
            break;
}
cputs(current_day);
cputs(" ");
i = atoi(current_month);
switch ( i ) {
    case 1: cputs("Jan. ");
            break;
    case 2: cputs("Feb. ");
            break;
    case 3: cputs("Mar. ");
            break;
    case 4: cputs("Apr. ");
            break;
    case 5: cputs("May ");
            break;
    case 6: cputs("June ");
            break;
    case 7: cputs("July. ");
            break;
    case 8: cputs("Aug. ");
            break;
    case 9: cputs("Sep. ");
            break;
    case 10: cputs("Oct. ");
            break;
    case 11: cputs("Nov. ");
            break;
    case 12: cputs("Dec. ");
            break;
}
strcpy(time_copy, display_time);
temp = strtok(time_copy, " ");
strcpy(ho,temp);
temp = strtok(NULL, " ");
strcpy(min,temp);
hour = atoi(ho);
minute = atoi(min);
if( (minute < 30) && (minute != 0) )
    minute = 0;
else
    if( minute >= 30 )
        minute = 30;
for( i = 4; i < 20; i += 3 )
{
    itoa(hour, ho, 10);
    if( hour < 10 )
    {
        strcpy(time_copy, "0");
        strcat(time_copy, ho);
    }
    else
    {
        if( hour == 24 )
            strcpy(time_copy, "00");
        else
            strcpy(time_copy, ho);
        if( minute == 0 )
            strcat(time_copy, ":00");
        else
            strcat(time_copy, ":30");
        if( strcmp(cappointment->end_time, time_copy) == 0 )
            textattr( LIGHTGRAY + 'BLUE'<<4);
        gotoxy(4,1);
        cputs(time_copy);
    }
}

```

```

cputs("
gotoxy(4,1+1);
cputs("
minute += 30;
if( minute > 30 )
{
    minute = 0;
    hour += 1;
}
if( hour == 24 )
    hour = 0;
} /* end for loop */
fill_comment(time_copy); /* fill appointments comments */
make_day_flash(0);
flash_day = atoi(current_day);
make_day_flash(1);
}

/*****
/* highlighth_day: use to do the actual highlighth of a date in the month */
/* window.
*****/

void highlighth_day ( int id )
{
    int x, y;
    div_t d;
    char c[3];

    if( (id == flash_day) && (into == select) ) /* current day flashing day */
        textattr(WHITE + (LIGHTGRAY << 4) + BLINK);
    else
        textattr(WHITE + (LIGHTGRAY << 4));
    if( (mode == PLANNING) && (select == 2) )
    {
        window(38,2,66,18);
        x = id + next_month_offset;
        y = id + next_month_offset - 1;
    }
    else
    {
        window(4,2,32,18); /* go to the month window */
        x = id + month_offset; /* calculate column */
        y = id + month_offset - 1; /* calculate row */
    }
    while( x > 7 ) x -= 7;
    x = 2 + ((x-1) * 4);
    d = div(y,7);
    y = 6 + (d.quot * 2);
    itoa(id, c, 10);
    gotoxy(x,y);
    cputs(c);
}

/*****
/* set_appointment_in_month: sets the days with appointment(s) to
/* white text on lightgray in the month window. It scans the agenda
/* for days, in the displayed month, with appointments and highlighth
/* them in the calendar.
*****/

void set_appointment_in_month(void)
{
    PYears cy;
    PMonths cm;
    PDays cd;
    int id, mm, iy, save;
    char temp[3], yea[5], mon[3];

    save = select;
    select = 1; /* always do the first month */
    cyear = Agenda;
    while( cyear != NULL )
        if( strcmp(cyear->yyyy, current_year) == 0 ) break;
    else cyear = cyear->next;
    if( cyear != NULL )
    {
        cmonth = cyear->month set;
        while( cmonth != NULL )
            if( strcmp(cmonth->mm, current_month) == 0 ) break;
        else cmonth = cmonth->next;
        if( cmonth != NULL )
        {
            cday = cmonth->day set;
            while( cday != NULL )

```



```

        {
            id = atoi(cday->id);
            highlight day(id);
            cday = cday->next;
        }
    }
}

if( mode == PLANNING ) /* then do the second month */
{
    select = 2;
    mm = atoi(current_month) + 1;
    if( mm == 13 )
    {
        mm = 1;
        yy = atoi(current_year) + 1;
        itoa(yy, yea, 10);
    }
    else
        strcpy(yea, current_year);
    itoa(mm, mon, 10);
    cy = Agenda;
    while( cy != NULL )
        if( strcmp(cy->yyyy, yea) == 0 ) break;
    else cy = cy->next;
    if( cy != NULL )
    {
        cm = cy->month set;
        while( cm != NULL )
            if( strcmp(cm->mm, mon) == 0 ) break;
        else cm = cm->next;
        if( cm != NULL )
        {
            cd = cm->day set;
            while( cd != NULL )
            {
                id = atoi(cd->id);
                highlight day(id);
                cd = cd->next;
            }
        }
    }
}

select = save; /* return select to its original value */
}

/*****
/* fill_function_window : fill a function window (delete and modify)
/* with the data of the current appointment. */
*****/

void fill_function_window (void)
{
    char *temp;

    window(5,6,75,15);
    if( (cyear == NULL) || (cmonth == NULL) ||
        (cday == NULL) || (cappointment == NULL) )
        return;
    textattr(LIGHTGRAY + (BLACK <<4));
    gotoxy(8,5);
    cputs(cappointment->start_time);
    gotoxy(17,5);
    cputs(cappointment->end_time);
    gotoxy(25,5);
    if( (temp = strchr(cappointment->comment, '\n')) != NULL )
        *temp = ' ';
    cputs(cappointment->comment);
}

/*****
                                     /*****
                                     /* Window Execution Module code */
                                     /*-----*/
*****/

int window_em ( int service, char *parameter )
{
    int id;

    switch ( service ) {
        case INITIALIZE: init_agenda_Windows();
                        return(1);

        case FILLMONTH:  fill_month_window();
                        return(1);
    }
}

```

```
case FILLDAY:    fill_day_window();
                return(1);

case OPEN_ADD:
case OPEN_DELETE:
case OPEN_MODIFY: open_func:on_window(service);
                return(1);

case COORD_TO_EVENT: return(coordinate_to_event(parameter));

case CLOSE_WINDOW: close_window();
                return(1);

case HIGHLIGHT: highlight_box(current_box, ON);
                if( current_box == 1 )
                    highlight_box(3, OFF);
                else
                    highlight_box((current_box - 1), OFF);
                return(1);

case SETAPP: set_appointment_in_month();
                return(1);

case FILL_COM: fill_function_window();
                return(1);

case OPEN_PASS:
case OPEN_NEW_PASS:
case OPEN_NOPASS: open_password_window(service);
                return(1);

case WRITELN: message_print(parameter);
                return(1);

case OPEN_MESS: open_message_window();
                return(1);

case TWO_MONTH: init_two_month_windows();
                return(1);

case FLASH:    id = atoi(parameter);
                make_day_flash(id);
                return(1);

default: return(NOSERVICE);
}
}
```

APPENDIX A

Personal Agenda code listing

208

```

/*****
 * file: WINDLIB.C */
*****/

/* This file contains library routine for windows. It can be considered */
/* as the possible functions/procedures that can be on the window object.*/

#include <stdlib.h>
#include <string.h>
#include <conio.h>
#include <dos.h>

/*****
 * window constants */
*****/

#define LTCORNER      0xc9
#define RTCORNER      0xbb
#define LBCORNER      0xc8
#define RBCORNER      0xbc
#define HORIZONTAL    0xcd
#define VERTICAL       0xba
#define TOPRIGHT      0xda
#define TOPLEFT        0xbf
#define BOTTOMRIGHT     0xd9
#define BOTTOMLEFT      0xc0
#define SINGLE_VERT    0xb3
#define SINGLE_HORIZ    0xc4
#define SEPARATOR      ".. /* parameters separators */

/*****
 * Window object global variables definition */
*****/

int window_number = 0;
char screen[10][(25 * 80 * 2)]; /* 10 buf for 25 lines x 80 cols */
short int windows[10][4]; /* up to 10 windows x,y coords */

/*****
 * PROCEDURE: save_screen( x1, y1, x2, y2, buffer )
 * Saves an area of the screen delimited by the provided coordinates
 * The screen is saved in the global variable screen[]. This buffer
 * is large enough to save the entire screen.
 *****/

void save_screen ( int x1, int y1, int x2, int y2 , char *buffer )
{
    if( !gettext( x1, y1, x2, y2, buffer) )
    {
        gotoxy(1,25);
        sound(125);
        delay(500);
        nosound();
        window(1,25,26,25);
        textattr(WHITE + (RED<<4));
        clrscr();
        puts("SCREEN SAVE ERROR");
    }
}

/*****
 * PROCEDURE: close_window(void). Closes the last opened window.
 *****/

void close_window (void)
{
    window_number--;
    puttext( windows[window_number][0],
             windows[window_number][1],
             windows[window_number][2],
             windows[window_number][3],
             screen[window_number]);
}

/*****
 * PROCEDURE: open_window( x1, y1, x2, y3, bkgnd, frame, title)
 * Creates a window of specified size, and puts a frame of double
 * lines around it of the frame is true. The window title is put
 * on the first line below the frame only if it fits in the window.
 * If no Title is present (i.e. title == NUL) none is printed.
 *****/

void open_window (int x1, int y1, int x2, int y2,

```

```

        int bkgrnd, int frame, char *title )
{
    div_t temp;
    int i, j;

    save_screen(x1,y1,x2,y2,screen[window_number]);
    textattr(WHITE + (bkgrnd<<4));
    window(x1, y1, x2, y2);
    clrscr();
    window(x1, y1, x2, y2+1);
    if( frame )
    {
        gotoxy(1,1);
        putch(LTCORNER);
        gotoxy((x2-x1),1);
        putch(RTCORNER);
        gotoxy(1,(y2-y1));
        putch(LBCORNER);
        gotoxy((x2-x1),(y2-y1));
        putch(RBCORNER);
        for( i = 2; i < (x2-x1); i++ )
        {
            gotoxy(i,1);
            putch(HORIZONTAL);
            gotoxy(i,(y2-y1));
            putch(HORIZONTAL);
        }
        for( i = 2; i < (y2-y1); i++ )
        {
            gotoxy(1,i);
            putch(VERTICAL);
            gotoxy((x2-x1),i);
            putch(VERTICAL);
        }
    }
    temp = div( (x2 - x1 + 2), 2 );
    i = temp.quot;
    j = strlen( title );
    if( (j > 0) && (j < (x2 - x1 - 2)) )
    {
        temp = div( j, 2 );
        j = temp.quot;
        i -= j;
        gotoxy( i, 2 );
        cputs(title);
    }
    windows[window_number][0] = x1;
    windows[window_number][1] = y1;
    windows[window_number][2] = x2;
    windows[window_number][3] = y2;
    window_number++;
}

/*-----*/
/* PROCEDURE: text_box( x, y, length, title) */
/* Makes a text box of specified length, with a frame around and a */
/* frame around. Text boxes are one line high, but require four */
/* lines to create, three when there are no title. The title is */
/* positioned at the top left of the text box. */
/*-----*/

void text_box ( int x, int y, int length, char *title )
{
    int i;

    gotoxy(x-1,y-1);
    putch(TOPRIGHT);
    for( i = 0; i < length; i++ )
        putch(SINGLE_HORIZ );
    putch(TOPLEFT);
    gotoxy(x-1,y+1);
    putch(BOTTOMLEFT);
    for( i = 0; i < length; i++ )
        putch(SINGLE_HORIZ);
    putch(BOTTOMRIGHT);
    gotoxy(x-1,y);
    putch(SINGLE_VERT);
    gotoxy(x+length,y);
    putch(SINGLE_VERT);
    if( strlen( title ) > 0 )
    {
        gotoxy(x,y-2);
        cputs(title);
    }
    textattr( LIGHTGRAY + (BLACK<<4) );
}

```

```
gotoxy(x,y);
for(i = 0; i < length; i++)
    putchar(' ');
}

/*.....*/
/* PROCEDURE highlight_box( int boxnum, int mode) */
/* Change the text box boxnum from low to high intensity, or high */
/* to low if mode = OFF. */

void highlight_box ( int boxnum, int mode )
{
    int x, y, len, i;

    y = 5;
    switch ( boxnum ) {
        case 1: x = 8;
                len = 5;
                break;
        case 2: x = 17;
                len = 5;
                break;
        case 3: x = 25;
                len = 36;
                break;
        default: return;
    }
    if( mode )
        textattr(WHITE + (BLUE << 4));
    else
        textattr(LIGHTGRAY + (BLUE << 4));
    gotoxy(x-1,y-1);
    putchar(TCPRIGHT);
    for( i = 0; i < len; i++ )
        putchar(SINGLE_HORIZ);
    putchar(TOPLEFT);
    gotoxy(x-1,y+1);
    putchar(BOTTOMLEFT);
    for( i = 0; i < len; i++ )
        putchar(SINGLE_HORIZ);
    putchar(BOTTOMRIGHT);
    gotoxy(x-1,y);
    putchar(SINGLE_VERT);
    gotoxy((x+len),y);
    putchar(SINGLE_VERT);
}
```

END

'20-07-94

FIN