Re-engineering Software for Integration using Computer Aided Software Engineering

by

Thierry Le Gal

Thesis submitted to the Faculty of the
Virginia Polytechnic Institute and State University
in partial fulfillment of the requirements for the degree of
Master of Science
in
Mechanical Engineering

APPROVED:

[Signature]
Dr. A. Myklebust, Chairman

[Signature]
Dr. S. Jayaram

[Signature]
Dr. R. Mahan

July 7, 1992
Blacksburg, Virginia
Re-engineering Software for Integration using
Computer Aided Software Engineering

by

Thierry Le Gal

Dr. A. Myklebust, Chairman

Mechanical Engineering

(ABSTRACT)

This research addresses the problem of integrating software at a high level of integration (analysis or design level) using Computer Aided Software Engineering. The assumption is made that the source code of the two applications to be integrated is available in language C or FORTRAN 77. Following the research done by Sandra Pennington (Penn91) and Michele Grieshaber (Grie91) on this subject, a new analysis has been led. Sandra Pennington (Penn91) conceived and proposed an "Integration CASE workbench" to support an integrator in his task of integrating two software.

The Integration Source Code Analyzer (ISCA), one of the components of the Integration CASE workbench has been designed and implemented as well as some utilities and features supporting the integration of software. Relevant information is extracted from the source code by a parsing tool. The input to this tool is the source code of the software to be integrated and their call trees generated by the Teamwork/C Rev reverse engineering toolkit. A database of the relevant information is created and can be consulted via the ISCA Display Manager developed using the Motif window manager. The Teamwork CASE workbench provides an interactive environment to lead the analysis and the design of software, automating standard structured methodologies in software development. Some features have been developed to extend the Teamwork capabilities and to import information into its environment. Some new utilities to help the integrator visualize a structure chart around a selected module have been developed and included in the environment using Teamwork User Menus capabilities. The parsing tool embedded in the environment can also be used as a stand-alone tool to lead re-engineering operations on software.
Acknowledgements

Funding for this research was provided by IBM Corporation under the grant no. EGP-1189. I would like to thank Dr. Arvid Myklebust, director of the CAD Laboratory at Virginia Polytechnic Institute and State University, for providing me the opportunity and the means to work on a Master’s thesis. I appreciate Dr. Sankar Jayaram and Dr. Robert Mahan for taking time to be on my graduate committee.

I would like to thank Carol Lena Terry who has been helping me on this project for six months as a part-time analyst programmer. I appreciated her professional competence as well as her human qualities. I would like to acknowledge my friends and colleagues who have been working in the VPI CAD laboratory during those two years. I wish them the best. In particular I thank Steve Fleming and Dr Wenhsyong Lin for their friendship. I also thank our devoted program support technician, Marti Altizer as well as our devoted secretary, Dee Meier.

I greatly appreciated the services offered by the CADRE technical support hot line. My numerous calls have always been processed with efficiency. The CADRE technical support engineers contribute to give a very positive image of their company.

Finally, but not the least, I would like to thank Dr. Grynszpan from l’Ecole Nationale Supérieur d’Arts et Métiers of Paris, without whom this thesis would have never existed.
Table of Contents

1.0 Introduction .......................................................... 1
1.1 Problem definition .................................................. 1
1.2 Organization of this thesis ....................................... 2

2.0 Research objectives ................................................ 4

3.0 Literature review .................................................... 6
3.1 Integration in CASE ................................................ 6
3.2 Software integration ............................................... 8
3.3 Re-engineering ....................................................... 10

4.0 Computer Aided Software Engineering ........................... 12
4.1 Introduction to CASE .............................................. 12
  4.1.1 CASE definition .............................................. 12
  4.1.2 CASE goals .................................................. 12
  4.1.3 CASE history ............................................... 13
  4.1.4 CASE benefits ............................................. 15
  4.1.5 Required capabilities for a CASE system .................. 16
  4.1.6 Software Engineering ....................................... 16
  4.1.7 Characteristics of Software Automation ................... 17
  4.1.8 Diagrams which support a software engineering approach 17

4.2 CASE Software Development Environment ....................... 19
  4.2.1 CASE Tools ................................................. 19
  4.2.2 CASE Toolkit ............................................... 20
  4.2.3 CASE Workbench ......................................... 21

4.3 CASE Repository Concept ......................................... 23

4.4 CASE Methodologies ............................................... 24
  4.4.1 Introduction ................................................ 24
  4.4.2 Evolution of CASE methodologies .......................... 24
List of Illustrations

Figure 1. The Integration CASE workbench (Penn91) (Grie91). .......................... 5
Figure 2. Software life cycle. ......................................................... 17
Figure 3. Components of a CASE workbench. ....................................... 22
Figure 4. CASE repository. .......................................................... 23
Figure 5. Data flow diagram. ......................................................... 27
Figure 6. Structure chart. ............................................................. 29
Figure 7. Warnier-Orr diagram. ...................................................... 31
Figure 8. Entity-relationship diagram. ............................................. 32
Figure 9. CADRE's call tree description. ......................................... 45
Figure 10. Actual call tree description. ............................................ 46
Figure 11. A call tree sample in Teamwork. ....................................... 47
Figure 12. Structure chart for the parsing tool. ................................... 49
Figure 13. Lyza.awk algorithm. ..................................................... 50
Figure 14. Format of the file raw.data generated by the parser. ............... 52
Figure 15. Data structure. ........................................................... 55
Figure 16. The ISCA interface. ..................................................... 59
Figure 17. The integrator environment. ............................................ 63
Figure 18. Structure chart for build_ctree_fct_level application. .......... 66
Figure 19. Description of the two major processes. ............................. 67
Figure 20. appl1.c source code. .................................................... 72
Figure 21. appl2.c source code. .................................................... 73
Figure 22. raw.data.appl1 flat file. ................................................. 77
Figure 23. raw.data.appl2 flat file. ................................................. 78
Figure 24. appl1.c Context-Diagram. .............................................. 79
Figure 25. appl1.c data flow diagram 0. ......................................... 80
Figure 26. appl1.c process specification 1. ....................................... 81
Figure 27. appl2.c Context-Diagram. .............................................. 82
Figure 28. appli2.c data flow diagram 0. ............................................... 83
Figure 29. appli12.c Context-Diagram. ............................................... 84
Figure 30. appli12.c data flow diagram 0. ............................................... 85
Figure 31. appli12.c structure chart main. ............................................... 86
Figure 32. appli12.c source code. ........................................................... 87
1.0 Introduction

1.1 Problem definition

Computer Aided Software Engineering (CASE) systems offer an automation of the software development life cycle. Developers can successively analyze their application, design it and generate the corresponding code using a CASE system. The application code can even be reverse engineered using a CASE system. CASE improves considerably the efficiency of large teams of developers working on a single project by improving the quality of their work and improving the quality of the software produced.

As more and more different software become available to companies, the need to exchange information between them is increasing. Quite often, a model designed on a CAD/CAM package has to be passed to another package to perform a finite element analysis on it. This is only one example among many. The integration of dissimilar software in order to allow this exchange of information between them becomes a necessity.

How can a CASE system help the integration of CAD/CAM software or any software? What environment needs to be provided to an integrator to facilitate the integration of different software? These are the issues addressed by this research. An integrator will no longer have to sit at his desk and browse two listings containing thousands of lines of code, trying to figure out where the necessary data are or where the pertinent functions regarding the integration are. Providing an environment to support the integration of software is the objective of this research.
1.2 Organization of this thesis

Required information is provided to the reader to bring him progressively to a good understanding of the area of this research. The proposed solutions have been explained in detail.

Chapter One gives a global description of the problem and restates it in the current economic environment.

Chapter Two defines with precision the objectives as well as the functional specifications.

A literature review revealing the issues around CASE is presented in chapter Three.

Chapter Four gives the reader an understanding of what CASE is. The information given in this chapter is the result of an exhaustive literature review. CASE is a very "hot and controversial" subject generating many publications. A theoretical background on CASE is provided to facilitate the understanding of the revolution going on in software development.

The Teamwork CASE system developed by Cadre Inc. is integrated in the environment. Despite the complexity of such a CASE system, there was no need to include technical information about Teamwork in this thesis. However, the required information to understand the capabilities of this product are provided. Some detailed sources of information are also referenced in chapter Eight. It must be noticed that this research constantly took place at the edge of the latest technologies in CASE: Teamwork 4.0.1 version as well as the Teamwork/C Rev, Teamwork/FORTRAN Rev and Teamwork C source builder were released less than five months before this writing.

The Integration Source Code Analyzer (ISCA) is described in chapter Five. The information system as well as the ISCA display manager are defined and explained in detail.

Chapter Six describes a simulation of the integration of two programs and provides an explanation of the successive tasks. The resources available to the integrator are also described.
The analysis of the results as well as some recommendations are expressed in chapter Seven.

The source code for trimming the call tree to generate a view of the structure chart around or below a given process, for extracting information from the call tree and the source code, for attaching a note to a Teamwork object, for creating and loading the database, for creating the ISCA display manager and retrieving information from the database, can be found in the appendices.
2.0 Research objectives

The word integration defines here the ability to share some data between software. To facilitate this integration, an Integration Source Code Analyzer (ISCA) will be developed and embedded in the Integration CASE workbench. The ISCA will create a database of valuable information about a given set of software modules and make it available to the integrator via an interactive Display Manager. Some tools to communicate between CADRE’s Teamwork Computer Aided Software Engineering (CASE) workbench and the ISCA Display Manager will also be developed.

Two major objectives have been defined:

- Allow the use of the Teamwork CASE workbench to lead the analysis of the software to be integrated and design the integration functions to be added.

- Provide in an integrated environment, all the relevant information concerning the applications to be analyzed in order to design the integration functions.

Sandra Pennington (Penn91) and Michele Grieshaber (Grie91) conceived the complete environment in which the ISCA and the Teamwork CASE workbench will be included. The sketch of the environment is reproduced in Figure 1 on page 5.
Figure 1. The Integration CASE workbench (Penn91) (Grie91).
3.0 Literature review

Chapter Two clearly defines the goals in providing some tools embedded in the Integration CASE workbench to facilitate the integration of different software. The three following fields have successively been investigated in the literature to depict the “state of the art” in this area. Those three fields are: integration in CASE, software integration (directly related with this research) and finally the domain of re-engineering. Those three parallel investigations turned out to provide many ideas and complementary information. They fully benefitted the understanding of the problems to be solved and largely helped the elaboration of a solution.

3.1 Integration in CASE

Although this domain is not directly related to this research, it provided much useful information and helped in defining and understanding the concept of integration.

Ronald J. Norman and Minder Chen (Norm92) insist on the necessity for an integrated CASE environment, necessity driven by the demand for ever faster development of highly integrated, very complicated strategic information systems. Going even further, they affirm that although CASE technology has made significant advances recently, its potential is limited by integration difficulties. They introduce a technical framework for an integrated CASE environment based on the reference model developed by the National Institute of Standards and Technology, and the European Computer Manufacturers Association. They propose an organizational framework that places the technology in the context of information-systems development and management at the enterprise, project, team and individual levels. A description is also given of the three forms of integration (data integration, control integration, presentation integration) defined in the NIST/ECMA reference model.
In another very recent paper, Alan Brown and John A. McDermd (Brow92) outline Integrated Project Support Environment's shortcomings and advocate integration at higher levels - semantic and method - to improve productivity and quality. The authors point out that integration can manifest itself in many ways within the development environment and they define the following key aspects of integration:

- Interface integration
- Process integration
- Tool integration
- Team integration
- Management integration

Alan Brown and John A. McDermd recall that most IPSEs have only tool integration and present a proposed classification of tool-integration levels in an IPSE. Many other experts confirm their statement and focus their research only in this field.

For example, Ian Thomas and Brian Nejmeh (Thom92) define what they mean by tool integration in software development as they consider that current definitions are not precise enough. They define tools integration by identifying its relationships with other tools in the environment and the properties that characterize those relationship. They identify the goals of tool integration and propose some questions that establish what information is needed to know that these goals have been reached.

Every expert feels the need to define what integration can mean in a CASE environment. Antony Wasserman (Wass90) for example, has identified five kinds of integration which are:

- Platform integration, concerned with framework services,
- Presentation integration, concerned with user interaction,
- Data integration, concerned with the use of data by tools,
- Process integration, concerned with the role of tools in the software process.

Aware of the difficulties to define the word "integration" in a CASE environment, the existent literature has been consulted to figure out how difficult the integration of software was to define, and how much work had been done in this area.
3.2 Software integration

Although the necessity to integrate the provided tools will have to be faced, the main concern is to define what is hidden behind the expression "integration of dissimilar software".

Under the same grant as the one accorded by IBM for this project, Sandra Pennington (Penn91) investigated the current state of integration of CAD/CAM software among twenty six Fortune five hundred companies. She focussed her survey on the following points:

- Type of CAD and CAM software currently used
- Level of integration possessed by these systems
- Type of integration used
- Type of information passed between systems
- Type of information needed to be shared for a good integration

The survey revealed that a majority of companies connect application programs and CAD/CAM software systems through data passing interfaces and translators. Seventy seven percent of the respondents were using IGES (Initial Graphics Exchange Specification). In absence of automatic methods, sixty five percent of the companies re-enter data by hand and often recreate geometry. Sandra Pennington realized that the word "integration" had a different meaning for most people and came out with the four following definitions:

- **rigidly-connected interfacing**: application1 is modified to produce an output file which is the same as the one expected as input for application2.
- **rigidly-connected integration**: two applications utilize the same common data and ideally appear as one application in which the original applications are accessed as functions. A common user interface is also a desired part of this method.
- **freely-connected interfacing**: requires the translation of data from application1 into a standard format for passing and then an additional translation from standard format into the format used by application2.
- **freely-connected integration**: both programs communicate with a common database but a DBMS manages the integration of shared data.
The author assumed in her work that the expression "integration of CAD/CAM software" was synonym of "data integration between CAD/CAM software". This is also the definition given to this expression for this research project. She presented the requirements and a detail design of a CASE integration toolkit. A new integration model, referred to as the High-level Autonomous Integration Model (HAIM) is also described in her work. The autonomous integration model allows a high-level of integration than is normally considered through the utilization of the X Window System and database methods. This model includes the transfer of necessary data as well as simultaneous execution of application programs and the ability to share menus and functions between applications.

In parallel, Ronald J. Norman and Minder Chen (Norm92) recall in their paper the following four information sharing methods defined in the IEEE draft standard for tool interconnection:

- Direct transfer
- File-based transfer
- Communication-based transfer
- Repository-based transfer

Although defined in the context of tool interconnection, it appeared that these definitions could fully benefit this research.

In a very close field and under the grant accorded by IBM for this research, Michele Grieshaber (Grie91) designed and implemented a multiplatform integration system, namely the "Network Environment Integration System Enabler" (NEISE), which provides facilities to integrate CAD/CAM software in a network environment. The NEISE is one of the tools included in the Integration CASE workbench presented in Figure 1 on page 5. It consists of an integration server at the center of the integration system which manages the exchange of data among the integration clients. To use the distributed integration solution, the integrator customizes portions of the structure charts, data dictionary, and module specifications contained in the workbench according to the needs of the applications programs and generate C-source code defining the integration system. The structure charts of the NEISE have been designed on the CADRE CASE workbench using Teamwork/SD, a toolkit for designing software applications using structured design techniques.
3.3 Re-engineering

Before designing the ISCA and using Teamwork/C Rev, a reverse engineering tool developed by Cadre Inc., particular attention was paid to the latest developments in the domain of reverse engineering. John Burke (Burk91) felt the necessity to define the terms reverse engineering and re-engineering which are often confused.

- **Reverse Engineering** is the process of analyzing a system to identify its components and their interrelationships and create representations of the system in another form or at a higher level of abstraction.
- **Re-engineering** is the process of identifying what you have in order to help determine what you can do next. Re-engineering combines the skills of reverse engineering with forward engineering to applications.

According to these two definitions, this research is closely connected with re-engineering. Some tools will be provided to help the analysis of an application and enable the design of some new code to be embedded in the initial code in order to exchange some data with other CAD/CAM software.

Andrew Mahon (Maho91) reminds how the escalating maintenance costs associated with a tremendous burden presented by aging software applications have targeted re-engineering as a critical element of the development process and as a crucial differentiating factor in selling CASE products. He insists on the fact that re-engineering is in need of methodological support; no standards exist for the re-engineering process. He points out that trying to re-engineer some systems which have been coded without following any structured methodology may be very difficult, if not impossible. That is why he examines the selection of which system to re-engineer with a lot of attention.

R. N. Britcher (Brit90) discusses in his paper the techniques used to successively re-engineer the New York terminal approach control (TRACON) software. This project demonstrated that a twenty years old, real time system could be cost-effectively re-engineered while retaining the behavior of the applications.

More closely related to the field of this research, X. Ge and N. Prywes (Ge90) led a project to re-engineer real time programs using CASE. The original aspect of their work resides in the use of a CASE system to assist them in their re-engineering task. They used
a repository to catalogue and graphically document the programs to be re-engineered. They took advantage of the diagramming tools (structure diagrams, data flow diagrams, object oriented diagrams, dependency diagrams) offered by their CASE system to visualize the initial programs to be re-engineered.

Ronald J. Norman and Minder Chen (Norm92) extrapolated the future of reverse engineering in their paper. Their views will be used to conclude this brief overview of the latest literature on reverse engineering and re-engineering. They affirm that reverse-engineering tools are useful when development information is available only in a low-level form (e.g. code), but in the long run they will be less useful because systems developed in integrated CASE environments are maintained through high-level specifications.
4.0 Computer Aided Software Engineering

4.1 Introduction to CASE

4.1.1 CASE definition

Computer Aided Software Engineering (CASE) is the technology for automating software development and maintenance. Although much of the early focus on CASE technology has been centered around software tools, CASE is much broader than this. There are software, hardware, methodology and management aspects to CASE technology. The ultimate goal of CASE technology is to automate the entire software life cycle process with a set of integrated software tools. CASE can assist a software engineer in his work by providing partially automated help, but the software development process cannot be fully automated. CASE dramatically changes the way we build software systems by providing three key advancements:

1. An interactive development environment with rapid response time, dedicated resources and early error checking
2. Automation of many software development and maintenance tasks
3. Visual programming provided by a powerful graphics interface

4.1.2 CASE goals

In the simplest terms, CASE is an infrastructure that provides common services - for example, a database and a generic user-interface component - and a tool set that provides software-engineering functionality. Another way to look at a CASE system is from the point of view of the technologies it uses. Taking this approach, you could define a CASE system from a user interface, database, or knowledge based system perspective.
We prefer to look at a CASE system from the view that best lets us identify integration needs - namely, in terms of its basic requirements. A CASE system's goal is to support software development - specifically, to provide a software-engineering team with a productive and efficient environment so that engineers can produce high-quality software on time and within budget. Clearly this is the goal of any support mechanism, but a CASE system is more than just an operating system and a collection of tools. The difference is in its aim to achieve higher productivity, quality, and controllability levels than standard facilities can achieve.

The traditional software life cycle emphasizes the later phases of software development whereas, in the CASE software life cycle, more time is spent on analysis and design, and the coding phase is virtually eliminated.

4.1.3 CASE history

Edward Yourdon opened the CASE UPDATE conference in March 1991 with the following statements:

- The first generation of CASE products (1984-1986) were simple drawing tools used on an individual basis.
- The second generation of products (1987-1989) were mainly intended for PC/AT-type machines. They offered better drawing, some early code generation, limited integration, and could be shared by a few workstations.
- The era of the third generation has begun. These products are mainly aimed at 386-based machines and provide multi-tool integration through a shared repository and beginnings to attempts an 'enterprise' level of CASE.

As Ronald J. Norman and Minder Chen (Norm92) remind us, the tools and methods that have evolved into today's CASE environments were designed to support application development. Therefore, CASE has been significantly influenced by the characteristics of the applications being built at the time and the methods that existed for building these applications.

New applications drive the creation of new systems-development methods. Because these new development methods are complex, tools are developed to help us use them. These more powerful tools, in turn, make it easier to develop new applications and methods.
Applications. In the 1970s, most business applications were batch transaction-processing systems in third-generation languages. As database technology matured, more complete, data-intensive, on-line transaction-processing systems were developed. In the late 1970s, several decision-support systems that help users analyze data via interaction with decision models were built.

In the 1980s, Ada was created when the use of real-time software embedded in control-and-communication devices grew. In the mid 1980s, expert systems and knowledge-based applications received a lot of attention. And in the late 1980s, organizations began to use strategic information systems to stay competitive, and top managers began to use executive information systems to retrieve internal and external information through desktop workstations.

In the 1990s, these information systems must be integrated to span all business functions, organizational levels, and global locations. Building these systems will require combining several enabling technologies. The pressure to build these systems on time and the complexity of the emerging enabling technologies (like client-server architectures and graphical user interfaces) are the major driving forces behind the push for integrated CASE.

Tools. Systems development generates volumes of complicated development information that must be captured and analyzed. A CASE environment lets system developers document and model an information system from its initial user requirements through design and implementation and lets them apply tests for consistency, completeness, and conformance to standards. The use of CASE technologies is becoming a critical factor in the success of large-scale systems projects.

In the early 1970s, first-generation CASE tools (such as PSL/PSA) were generally mainframe and text based. Interest in these tools helped spark the development of structured methods, which led to the realization that automated tools were necessary to gather and help analyze the voluminous development information that the use of structured methods generated.

The advent of graphical user interfaces on PCs and workstations led to graphical, front-end tools to support structured methods. However, the information captured by
these early tools was stored in the tools themselves and was generally not transferable among tools.

Second-generation CASE tools developed in the early 1980s were designed mainly to support structured methods that use graphical notations, such as data flow diagrams for structured analysis and structure charts for structured design. They incorporate better analysis functions to enforce the method’s rules.

The detailed development information these tools capture is stored in a project dictionary to be shared by other CASE tools in the same environment. However, this kind of integration is limited to tools from the same vendor and, generally, data within the same project.

The repository-based CASE products that became available in the late 1980s offer enterprise-wide and project-level local repositories. The repository integrates a set of tool kits for planning, analysis, design, programming, testing, and maintenance. However, many of these products depend heavily on the method used and tend to support only certain types of application development.

The 1990s will be an era of systems integration through open systems, driven by the demand for faster development of highly integrated, very complicated strategic information systems. Integrated CASE environment empower information system organizations to deliver these systems on time and to migrate systems among open platforms.

4.1.4 CASE benefits

- Makes structured techniques practical
- Improves software quality through automated checking
- Makes prototyping practical
- Simplifies program maintenance
- Speeds up development process
- Allows developers to focus on the creative part of software development
- Encourages evolutionary and incremental development
- Enables reuse of software components
This list suggests that CASE is not an entirely new technology but is built on techniques and tools that have been proven to work in practice. CASE can be defined as a re-packaging of structured concepts and methodologies with a new twist. That new twist is automation.

4.1.5 Required capabilities for a CASE system

To provide full software support, a complete CASE system must have the following capabilities:

- Graphics capability
- Error checking
- Information repository
- Tightly integrated toolset
- Full life cycle coverage
- Prototyping support
- Automatic code generation
- Structured methodology support

4.1.6 Software Engineering

As Bauer explained, Software Engineering is "the establishment and use of sound engineering principles (methods) in order to economically obtain software that is reliable and works on real machines."

Software Engineering is characterized either by:

- A top-down approach for developing software programs
  A step-by-step development process that begins with the most general function view
  The decomposition of this view into subfunctions
  The repetition of this process
- An object-oriented approach

The result is a hierarchically structured modular program.
4.1.7 Characteristics of Software Automation

The basic idea behind CASE is to provide a set of well-integrated, laborsaving tools linking and automating all phases of the software life cycle represented in Figure 2 on page 17.

![Software life cycle diagram](image)

Figure 2. Software life cycle.

Traditional software technologies are of two types: manual methodologies and standalone tools. CASE technology is a combination of well-integrated tools and methodologies, with emphasis on automation of the software life cycle.

CASE makes manual structured methodologies practical to use by automating the drawing of structured diagrams and automating generation of system documentation.

4.1.8 Diagrams which support a software engineering approach

While different types of methodologies have been presented, there are basically four kinds of diagrams which can adequately represent any software system:

- Data flow diagram: shows top-level functional processes in a system and data flow from acquisition through processing and eventual output
- Hierarchical tree-structured diagram: shows the relationship between program modules and shows hierarchical data structures
- Detailed procedural logic diagram: shows detailed program logic (sequence, selection, loop)
- Screen and report layout: shows design of the user interface
4.2 CASE Software Development Environment

4.2.1 CASE Tools

Definition

A CASE tool assists a software engineer in a specific phase of the software life cycle. The meaning of the term CASE tools covers tools for all aspects of systems design, and also systems construction.

What is good about CASE tools?

- Give aids for documenting results (more time can be spent on creative work, instead of tedious drawing work)
- Some CASE tools have facilities for code generation and some also support limited program code generation (e.g. automatic generation of database schemes)
- Standardization of methodologies and techniques used in the organization

What is bad about CASE tools?

- The users are limited to the techniques/methodologies supported by the available tools

Examples of CASE tool tasks

- Diagramming tools for drawing structured diagrams and creating pictorial system specifications
- Screen and report painters for creating system specifications and simple prototyping
- Dictionaries, database management systems and reporting facilities for storing, reporting, and querying technical and project management system information
- Specification checking tools to automatically detect incomplete, syntactically incorrect and inconsistent system specifications
- Code generators to generate executable code automatically from pictorial system specifications
- Documentation generators to produce technical and user system documentation required by structured techniques
CASE tools cover many types of software development and maintenance tasks.

Although tools are an important part of CASE, the CASE technology consists of more than software tools. CASE is a redefinition of the entire software environment.

4.2.2 CASE Toolkit

Definition

A CASE toolkit is a set of integrated CASE tools that have been designed to work together and to automate (or partially automate) a phase of the software life cycle.

The five basic phases of the software life cycle

- Analysis (defines the requirements for a solution)
- Design (plans how the system will be built)
- Code generation
- Tests
- Maintenance

Analysis toolkits automate the creation of the analysis specification, which lists the system requirements. An analysis toolkit usually includes structured diagramming tools for the generation of data flow diagrams and other necessary diagrams, a repository and a specification checker. Analysis toolkits were the first CASE tools to become available commercially.

Design toolkits support the creation of hierarchical tree-structured diagrams and procedural logic diagrams. They also provide tools and diagrams for the logical and physical design databases and files, as well as logical data modeling.

The programming phase of the software development life cycle is supported by tools that assist in the actual coding of the design system. The most important tool in this toolkit is a code generator. The code generator provides automatic generation of source code from program design diagrams and information. This automation increases the reliability of the produced code and reduces the amount of development time required. Integrated CASE tools in this area were the last to be commercially available and are still at a relatively low level.
The testing phase of the software development life cycle can be assisted by CASE tools for all phases. Specification checkers at the analysis and design phases help catch problems before they are generated in the code. Compilers and interpreters with interactive debugging facilities find problems at the coding level. Additionally, CASE tools which monitor performance, prepare test data and generate audit trails of all system documents, specifications, program code and test cases are usually available.

Maintenance tools currently dominate the CASE tools marketplace. The toolkits typically provide tools which correct, restructure, replace or enhance existing systems systematically. Large applications can be maintain more easily during their entire life. These toolkits are in widespread use in almost every type of organization and are required by most government agencies. Another tool which is a part of the maintenance toolkit is reverse engineering. This tool allows an existing program to be translated back into its design specification, including logical data models. Changes can now be made at the design level and the code automatically regenerated. Sophisticated tools of this type will revolutionize the CASE tool market but are currently not available except at a relatively low level.

4.2.3 CASE Workbench

Definition

A CASE workbench is a set of integrated CASE tools that have been designed to work together and to automate (or provide automated assistance for) the entire software life cycle, including project management, analysis, design, coding, testing and maintenance.

CASE workbenches differ from earlier programming environments in their breadth of coverage of the software life cycle.

Characteristics of a complete CASE workbench

- Graphics interface for drawing structured diagrams
- Information repository for storing and managing all software system information
- Highly integrated toolset sharing a common user interface
- Tools to assist every phase of the life cycle
- Prototyping tools
• Automatic code generation from design specifications
• Software life cycle methodology support with extensive checking built into the tools

Components of a CASE workbench

As represented in Figure 3, the basic components of a CASE workbench are: a front-end, a repository and a back-end.

![Components of a CASE workbench](image)

The front-end component corresponds to the early phases of the software life cycle: analysis and design.

The back-end component corresponds to the latter phases of the software life cycle: program implementation and maintenance.
4.3 CASE Repository Concept

The CASE repository is the real heart of the CASE workbench. Figure 4 shows the information contained by it: all system information, including technical information, project management information and relationships among the various system components.

![CASE Repository Diagram]

Figure 4. CASE repository.

In summary, the CASE repository is a central concept of the CASE technology. Specifications information stored in the CASE repository is used to generate system code and to maintain systems. This information can also be reused in the development of future systems.
4.4 CASE Methodologies

4.4.1 Introduction

The new idea of using charts and diagrams designed to reduce errors and improve the quality of completed code helped to impose a rigid structure. Structured methodologies help program designers in visualizing their design through the use of diagrams and schematics. This is why structured diagramming techniques are so important in CASE methodologies.

4.4.2 Evolution of CASE methodologies

Structured programming was one of the first efforts to develop a systematic method to support systems development. Because of high cost of correcting errors introduced early in the life cycle, structured techniques for design, analysis, and planning were subsequently developed.

In the mid to late 1980s, real-time systems design and object-oriented analysis and design methods were developed to amend the shortcomings of structured techniques. This period also witnessed the development of some interrelated methods and techniques (like information engineering) that took into account the entire life cycle. The rigor and the complexity of these methods dictated the use of CASE to apply them.

Because code generators and fourth-generation languages have somewhat simplified downstream activities, the development bottleneck has shifted to upstream activities, including systems planning, enterprise modeling, and requirements engineering.

The quality of upstream products is determined by how well systems personnel can get users and managers involved in development. Structured methods, developed in the 1970s, became popular in the 1980s, fueled by the rise of graphical notations and end-user orientation. However, many structured methods deal with only one or a few aspects of information-systems models, and critics say they are too imprecise and ambiguous.
Formal methods are more rigorous, but they are often not appreciated for communication between users and developers. Formal methods have complementary strengths and weaknesses; the possibility of integrating them is attractive.

Objective measures of productivity and quality are essential if we are to improve the development process. Measurement can be improved when process management is fully integrated in CASE environments because metrics can then be collected automatically. The field of software metrics will continue to grow in the 1990s, because using metrics in conjunction with tools and methods and applying statistical process control will help us better manage the development process.

Many current CASE tools, which were designed to support specific methods, make it practical to apply methods. The symbiosis of CASE tools with methods suggests that advancement in integrated CASE may rely on the development of methods that integrate many modeling perspectives within and across all life-cycle phases and across many application domains.

### 4.4.3 Structured diagramming techniques

Structured diagramming techniques are top-down and hierarchical. This means that they support a top-down, structured development approach.

They give a logical rather than a programming-oriented representation, which is more meaningful to users and nontechnical managers.

**Functions provided by the structured diagramming tools:**

- An aid to clear thinking and problem solving
- Precise and recorded communication between members of the development team, users and management
- Standardized representation of program architectural structure and data structure
- An aid to find program bugs
- An aid to analyze and understand existing programs
- An aid to change systems and programs (as they are built and during maintenance)
- Fast development with computer-aided programming
• A mechanism enabling users to review program requirements and design specifications
• A mechanism for automatic program documentation production

4.4.4 Structured analysis

Structured analysis is a top-down, functional decomposition to identify system requirements. The analysis process produces a system specification which serves as a blueprint for the system to be created. The set of tools which produces a structured specification consists of:

• Data flow diagrams
• Data dictionaries
• Process/control specifications

Data flow diagram

A data flow diagram depicts an application program from the point of view of the data. Data elements flow from process nodes to process nodes where they are modified. There is no notion of control flow in a data flow diagram, and for this reason, the data flow diagram best depicts the system as the end-user views it.

Flowcharts focus almost entirely on control flow instead of the data flow. Flowcharts usually say very little about data structures, in contrast to data flow diagrams.
Figure 5. Data flow diagram.
Data dictionaries

Data dictionaries are databases for the definitions of all data elements defined in a data flow diagram. These definitions consist of the components which make up the data item and the relationships between them. Structured analysis CASE tools provide a facility which maintains the data dictionary by adding new data flows to the data dictionary automatically when they are created.

Process/Control specifications

Process/Control specifications or minispecs describe the task performed by a single, bottom-level process node in a diagram. They are called minispecs because they document only one single process in the entire data flow diagram.

4.4.5 Structured design methodologies

Structured design methodologies guide the transformation of a data flow diagram into a structured design chart. CASE assists at this stage of software development by providing mechanical aids which help to translate detailed analysis requirements into design specifications for implementation.
Figure 6. Structure chart.
Two of the most common structured design methodologies are:

- Yourdon structured design (refer to (Your89))
- Jackson structured design

### 4.4.6 Data modeling methodologies

Data modeling assists a developer by allowing graphical modeling of data to be stored in the database. A data dictionary is used to organize all the data elements in the design of an application.

Two of the most common data modeling techniques are:

- Warnier-Orr diagrams
- Entity-Relationship diagrams

Moreover, these two techniques are of particular interest because they are common components of the analysis and design CASE tools.
Warnier-Orr diagrams

Warnier-Orr diagrams were invented by Jean-Dominique Warnier in France and later enhanced by Kenneth Orr in the United States. The principal use of Warnier-Orr diagrams is in designing data structures and file formats. Warnier-Orr diagrams are useful for both data structure and program architecture design.

Figure 7. Warnier-Orr diagram.
Entity-Relationship diagrams

Entity-Relationship diagrams are used to describe the relationships between data in an organization or in a conceptual model of a system or process. Entity-Relationship diagrams are well suited for information modeling and describing relationships between data elements.

Figure 8. Entity-relationship diagram.
4.5 Integration in CASE

4.5.1 History of integration

In the field of commercial data processing, pressure to improve the speed and quality of software production has been a driving force for many years. Initial improvements were made through the adoption of third generation languages such as COBOL and PL/1. Later, the use of database systems instead of conventional file-based techniques provided a greater level of independence between the application programs and the physical data structures. In the latest attempts to provide more powerful application languages, the data definition and manipulation languages of relational database systems have been augmented with facilities to provide easier end-user application development. As these languages manipulated sets of tuples of relations rather than records within files, they are often classified as the next generation of programming language, fourth generation languages (4GLs).

In effect, a 4GL provides an application development environment consisting of a relational database and application development language.

Hence, 4GLs can be seen as a limited form of IPSE in that they provide integrated support for the fast development of data processing applications.

4.5.2 Tool integration

What is tool integration in CASE?

Tools will be united by one common user interface with one common language to simplify their use. They will be designed not only to coexist, but also to know about each other and to call and feed each other automatically to accomplish the tasks put in front of them.

Obstacles to tool integration:

- Tendency to be stand-alone tools
- No standard interface to one another
• Highly dependent on a particular platform (operating system)

How should it appear to developers?

The CASE workbench tools should interact with each other in a consistent, intuitive way and should conform to a set of well understood standards. They should appear to the user to be cooperating with each other and they must be aware enough of each other not to duplicate functionality. Frameworks provide a structure to help in integrating CASE tools.

Why do CASE tools need to be integrated?

CASE tools are often written without knowledge of other tools with which they may need to interact. That is why the use of individual tools within a larger development context can be difficult.

The following problems are often experienced:

• Different development tools often overlap in their roles by duplicating effort.
• Tools designed to fit different development methods can often interfere with one another, produce inconsistent results or be totally incompatible.
• The complex relationships and dependencies which exist between data items are often lost or are difficult to determine as properties are created within the tools and are unavailable outside the tool, being hidden internally or transformed before output.
• There is no integrating structure which controls the ways in which a set of tools is allowed to interact with the data and the tools which individual users are allowed to invoke. As a result, maintaining data integrity is difficult.
• Current practice relies heavily on manual cooperation and communication to ensure that project members can work independently and yet as a team.

This list points out the need for CASE tools to be integrated in their management.
4.5.3 The IPSE approach

IPSE stands for Integrated Project Support Environment. The only clear distinction between CASE environments and IPSEs seems to be the application domains. While CASE environments address commercial data processing applications, IPSEs are primarily aimed at large-scale, embedded real-time systems, typically for use in military or process control environments. The emergence of CASE environments in commercial data processing can be seen as a direct parallel to the use of IPSEs in real-time embedded systems. Alan W. Brown and John A. McDermid (Brow92) have recently pointed out the main differences between CASE tools and IPSEs. He recognizes that they are evolving so that the distinctions are becoming less clear, but he enumerates three fundamental differences remaining:

- IPSEs are intended to support multiple methods and be open, while CASE tools support single methods and are not readily extensible.
- IPSEs are aimed at group work, while CASE tools are designed primarily for individual support.
- IPSEs have been used primarily in scientific and engineering applications, while CASE tools have been developed in more mainstream data-processing applications.

Having recognized the problems of integrating a set of individual tools to support the complete life cycle of a large software project, researchers began to focus their attention on the means of integration within a support environment, rather than between individual tools.

Integrated support has been provided by removing the data structuring and control facilities that are often duplicated in each tool and maintaining them at a central point.

Some research has led to the creation of an open environment. In this approach, the role of the IPSE is seen as the provision of an infrastructure into which tools can be embedded. It provides control of all data developed during the lifetime of a project by providing facilities accessible through a structured interface. Thus, facilities at this interface may include support for the structuring and storing of information for configuration and version control of data items, and for the sharing of this data among groups of users.
It is necessary to point out that analysis and design are viewed as the most critical software life cycle phases.
4.6 Application of CASE

After having been through the CASE theory, an idea of the way organizations are applying CASE must be given as well as the kind of problems they are running through.

In April of 1989, the working group “Experiences with the use of CASE tools” of the Dutch User Group of Structured Development Methodologies conducted an inquiry into the use of CASE tools among eight hundred and thirty four Dutch organizations. The purpose of this survey was to make an inventory of the experiences of Dutch CASE tool users.

Five major fields were surveyed:

- The characteristics of CASE tool users
- The selection criteria for CASE tools
- The implementation in organizations
- The actual usage of CASE tools and future expectations for the use of CASE tools

The result of the survey revealed that:

- CASE tool users were moderately positive about CASE tools which they use
- The appreciated aspects were:
  - Quality of diagrams
  - Correctness and consistency with regard to applied methods and techniques
  - User friendliness
  - Future potential
- Many users were unsatisfied with:
  - Poor interfaces with other software products
  - Lack of support for multi-user environments
  - Limited possibilities in adapting the tool to their own standards
- The main reason for purchasing these tools was to achieve quality improvement

CASE has evolved a lot since 1989. Companies are becoming much more familiar with its technologies. They are applying CASE more widely and their experience has been growing steadily. However this survey still represents a valuable picture from the user’s point of view. Despite many improvements, some arguments are still valid today.
Not everything goes smoothly when introducing CASE in an organization. For example, Edward Yourdon quoted from a survey the following problems in implementing CASE:

- High cost,
- Learning curve, the cost of training, and the time-to-implement,
- Integration of CASE into the existing tools and procedures,
- Assimilation of CASE into the existing culture,
- Necessity to have realistic expectations,
- Necessity to be prepared, to have a technical foundation on which to build,
- Necessity to start slow,
- Necessity to have the right tool for the job, and
- Uselessness of implementing CASE if the organization is not ready.

After these remarks, it is obvious that some guidelines have to be followed to lead to a successful CASE utilization.

**Guidelines for a successful CASE utilization**

The fifth annual international teamworkers conference will take place between June 28th - July 2nd, 1992 in Scottsdale, Arizona. It will be a very good opportunity for Teamwork users to share real experiences and knowledge with their fellows. A stage has now been reached where a company willing to stay at the front end of its software development business has experienced CASE tools for couple of years. Edward L. Knoll from Texas Instruments Incorporated is one of these experts whose experience in the application of analysis/design type CASE tool technology spans several years. He published interesting reflections about CASE in the Teamworkers Newsletter issued in fall 1991 (Knol91). Edward L. Knoll explains that many organizations are investing in CASE tools; especially, in those tools dealing with analysis and design modelling like Teamwork. This investment is made with the expectation of developing less expensive and better quality products. He points out that actual experiences by the users are definitely not uniform in regards to the effectiveness of these tools.

These organizations are encountering problems when integrating the CASE tools into the workplace. They invest in training for the methodology and the tool, but they discover that, while necessary, the training is not always sufficient. It can be difficult to get
users to use, or to continue to use CASE tools. If the tools are used, the anticipated cost savings and quality gains don’t seem to meet expectations, at least initially.

Edward L. Knoll published a paper explaining in further detail his opinions, viewpoints, and recommendations which he developed from his own experience. He considers that the problem seems to be that most new programs view their CASE tool like a new word processor or software compiler. If the tool is purchased, the appropriate hardware acquired, and some amount of training provided, the engineers will be able to use the tool and to be productive with that tool. Unfortunately, this is far from reality. Effective utilization of CASE seems to require changes in the program organization and development process.

His experience has been that most programs suffer from the same kinds of general problems when dealing with CASE. The most serious and most common problems have been organized into the following general categories:

1. CASE integration in the organization
2. Development process
3. Documentation generation

Edward L. Knoll insists on the fact that effective CASE utilization requires integration with the organization and the process, not just installation on the appropriate platform. Much of the frustration of new users stems from their attempting to utilize this technology in an organization that is not prepared to accept it or attempting to utilize this technology in a process that has not been adapted for its use. New programs must be prepared for an investment that goes beyond purchase of the tool and associated training. Time and resources need to be invested up-front to determine how the tool is going to be integrated into the development process and development environment.

To conclude his paper, Edward L. Knoll explains that upfront planning and preparation can not be stressed strongly enough. Several programs he has been on had to do extensive rework because:

1. the objectives were not clear,
2. nobody knew explicitly how the tool was to be used to achieve the objectives, or
3. the interfaces with other tools in the environment were not considered.
How to adapt and modify the development process for CASE utilization has not been completely characterized, however, there are some general recommendations that Edward L. Knoll makes.

1. Integrating CASE into the overall development environment is not simple or easy. If at all possible, get an experienced user to help integrate the tool into your development environment. Plan to scope additional resources and schedule to allow for this integration.

2. The environment inside the CASE tool is not as mature as the environment outside the tool. This can generate some resistance towards using the tool. This situation can be alleviated by having software technicians or computer literate clerks available on a part-time or full-time basis to help with model entry, model updates, or some of the more tedious tasks. However, the entire team must use the tool. If the tool is only used by a few individuals, the effort is likely to be late if it is successful at all.

3. The tool will not do everything you desire, or there will be certain aspects of it you will want or need to tailor. Scope additional resources and schedule to allow for tailoring of the tool environment. Attempt to include at least one person as part of your development team capable of performing this kind of work.

This paper gives a better understanding of the actual use of CASE tools in an organization and reveals that a successful CASE utilization is far from being easy.

Anybody willing to introduce CASE in an organization must be aware of the human resistance he will have to overcome:

- Senior management sees CASE as a strategic weapon. It is useless to try to convince them of the value of "value added" properties.
- Organizations that are willing to be innovators worry less about the costs.
- The technological imperative argument should be used when a competitive disadvantage can be shown without CASE.
- Middle management is emotionally involved with anything that affects their career. In addition to CASE, they worry about organizational change, loss of power, control, prestige, etc.
• Senior technicians have worries similar to middle management. Junior technicians are more willing to adapt to change. "Novices" are not aware that a revolution is underway.

• Pilot projects are important. Their size is critical. They should be staffed by enthusiastic volunteers who are well trained and supported.

4.7 Conclusion

Given the publicity and interest currently surrounding the subject of CASE, it is often forgotten that CASE tools are just that: a set of tools. To be effective, they must be understood and efficiently utilized. Even so, they still won't do everything. Organizations waiting for the perfect, easy-to-implement solution to all their software problems probably will be waiting for a very, very long time. CASE technology will never be a substitute for good project management.

So what will be the future of CASE? Tomorrow's complex, integrated applications will be developed using a combination of several enabling technologies (database and knowledge-based systems, object-oriented technology, and hypermedia). Today's CASE environments support application development in limited domains using one or two specific enabling technologies. Future integrated CASE environments will support a wider range of applications in an open-systems environment.
5.0 The Integration Source Code Analyzer (ISCA)

5.1 The ISCA Information System

5.1.1 Introduction

This chapter presents the fundamentals of the ISCA. The valuable information for an integrator are described in detail. A description of the call tree generated by the Teamwork/C Rev reverse engineering tool is provided since it will be used as input by the parsing tool. A detailed description of the output produced by the parsing tool as well as the description of the data structures used to store these information is also provided. Finally, the ISCA Display Manager is presented with a description of its capabilities.

Course of action

Given the location and name of a Unix "makefile" for an application, a database of relevant information can be built. It requires that all source code which is referenced in the makefile should be available. The database is built using a variety of existing tools. Using the Teamwork/C Rev reverse engineering tool, a call tree is produced for the application. The call tree inherently contains module names and the calling hierarchy as well as module types (regular, library, or global variable) and locations. The call tree gives an image of the source code at a higher level of abstraction. Unix shell scripts and the sed and awk parsing tools are employed to scan the call tree for relevant information (refer to the paragraph concerning the extraction of information from the call tree). The actual source code is referenced when necessary and a database is created.
5.1.2 Required information for the integrator

Two types of relevant information need to be provided to the integrator to get a more in depth knowledge of the application being analyzed:

- Individual module information
- Global variable information

Detailed description of the information maintained in the database for an application:

- Individual module information
  1. Name of module
  2. Kind of module - private or library
  3. Module type as defined in the source code
  4. Filename and line number where the module is defined
  5. Names and types of formal parameters to the module (these are extracted at the module definition)
  6. List of modules calling this module
  7. List of modules called by this module
  8. List of global variables accessed in the module
  9. Whether the module is an i/o module

- Global variable information
  1. Name of each global variable
  2. Type of each global variable
  3. Filename and line number where the global variable is defined
  4. List of modules accessing this global variable

The actual parameters at a function call are not extracted and stored in the database. If this is to be done, the line numbers of the function calls should be stored and associated with the actual parameters. Concerning the information about local variables accessed in a module, this has not been done up to now.

5.1.3 Definitions

A private module is a module coded by the developers of the application and for which the source code is available.
A library module is a module accessed by the current analyzed application and for which the source code is not available.

An i/o module is defined by the integrator himself. As it depends on the language used in the analyzed application, the integrator is given the possibility to define his own i/o library. For example, if the application is written in C language, the integrator can define the functions "printf", "fprintf", etc as i/o modules. If the application uses some graPHIGS calls, the function call "gputws" can also be added to the i/o library. The nature of an i/o function can be a file access, a database access, a screen access, an input device signal, etc. Any library module can be defined in the i/o library.

The possible modules are:

- Private module
- Regular library module
- i/o library module
- Global variable module

Their types can be of any type returned by a function or of any type taken by a variable (int, float, char, pointer, etc).

5.1.4 Extraction of information from the call tree

The extraction of some information from the call tree represented a critical step for this project. It required a perfect understanding of the format of call trees. The algorithms of the processes used to generate structure charts from a call tree had to be understood too. Some wrong information given by CADRE Inc. initially misled this analysis. The actual algorithm describing this logic has been established based on the one described by CADRE Inc. page 3-29 of Teamwork/C Rev user's guide release 4.0.1. The main error in the original algorithm conducted to define a global variable invoked by a function as being a library module. Figure 9 on page 45 and Figure 10 on page 46 reproduce these descriptions.
# Description of the call tree provided by CADRE Inc.
# Some comments have been added
#

line format: num tabs identifier: <space> description

<num>
  if <= level then next identifier is caller, level = num
  if > level then identifier is invoked by caller

<tabs>

<identifier>
  all characters until ":"

<space>

<description>
  if numeric
    --> Can be a function or a global variable invocation, or a library.

    Look for num line in call tree. If no definition found at that line for the identifier then this is a library module.
    If a definition is found then this is an invocation of the described module.
    --> Presented by CADRE but false in reality.
    In reality, scan the entire call tree to find a line with the same identifier having a non numeric description.
    If such a line is found, then this is an invocation of the described module else consider the identifier as a library.
  if "<":
    --> Never found in reality.
    This is a library module.
    --> This pattern is never found but if you insert a line looking like num tab identifier <space> <" it does not
    produce an error when running crev. The identifier is represented as a library on the SC.
    Note: a simpler way to define a library is to have a numeric description and no other lines in the call tree with the
    same identifier having a non numeric description (otherwise, it would be an invocation of that module).

  else [module description, file_description]
    module description
    read until comma, if text contains "(" then this is a plain module (function definition) (give the function type).
    else this is a data-only module (global variable definition) (give the type of the global variable)
    comma
    file_description
    read '<
    read filename
    read space
    read linenum
    read '>'

# Plain module:
  Used for function definitions, or for function references when the function does not call any other functions
  or library, and does not refer to global or static data

# Data only module:
  Used to declare or refer to global and static variables

# Library module:
  Used to reference functions or variables that are not defined in the current set of files.

Figure 9. CADRE's call tree description.
# Actual description of the call tree established by the author

line format: num tabs identifier: description

<num>
  if <= level then next identifier is caller, level = num
  if > level then identifier is invoked by caller
</tabs>

<identifier>
  all characters until ":" 
</space>

<description>
  if numeric
    → Can be a function or a global variable invocation, or a library. Scan the entire call tree to find a line with the same identifier having a non numeric description. If such a line is found, then this is an invocation of the described module else consider the identifier as a library.

  else [module_description, file_description]
    module_description
      read until comma, if text contains "(" then this is
      if text contains "(\" then this is a function definition
      (give the function type)
    else this is a global variable definition
      (give the type of the global variable)
    comma
    file_description
      read '<
      read filename
      read space
      read linenumber
      read '>'

# Library module:
Used to reference functions or variables that are not defined in the current set of files.

Figure 10. Actual call tree description.
To illustrate those algorithms, an example of a call tree is provided Figure 11.

```
1 buffer: char [], <calcget.c 31>
1 buffer_pointer: int, <calcget.c 32>
1 ungetch: void (), <calcget.c 47>
  buffer_pointer: 0
  fprintf: 0
  buffer: 0
1 getch: char (), <calcget.c 75>
  buffer_pointer: 0
  buffer: 0
1 getop: char (), <calemain.c 58>
  getch: 0
  ungetch: 0
1 main: void (), <calemain.c 125>
  printf: 0
  getop: 0
  push: 0
  atof: 0
  pop: 0
  fprintf: 0
  clear: 0
1 stack_pointer: int, <callestck.c 32>
1 stack_values: double [], <callestck.c 33>
1 clear: void (), <callestck.c 54>
  stack_pointer: 0
1 pop: operand_1(), <callestck.c 74>
  stack_pointer: 0
  stack_values: 0
  fprintf: 0
  clear: 0
1 push: operand_1(), <callestck.c 104>
  stack_pointer: 0
  stack_values: 0
  fprintf: 0
```

Figure 11. A call tree sample in Teamwork.
Knowing the mechanisms of the call tree interpretation, the following information is extracted from the call tree or/and from the source code:

- The module name, its kind (function, library or global variable), its type (void, int, float, etc.) and its location (file name including the path, line number).
- If the module is a function,
  - The global variables accessed by the function, their type
  - The parameters at the function definition (extracted from the source code), their type
  - The functions called by the function (excluding the global variables which need to be filtered from the call tree)
  - The functions calling this module.
- If the module is a function defined as a library in the call tree, the global variables accessed by this function, the function parameters at the function definition and the functions called by this function are not available and cannot be extracted. Therefore, they won’t be stored in the database. In this special case where the module is either a library or an i/o function defined by the user in the system_io file, the calling functions will be the only information extracted and stored in the database.

The processes realizing the extraction of the information use some Unix scripts, some C programs as well as the sed and awk parsing tools. Carol Lena Terry initially developed those processes. They have been completely modified since then and the processes extracting the parameters from the source code are the only ones which have been kept as originally designed. The source code is accessed using the module location contained in the call tree in order to extract the function parameters at the function definition.

These programs are included in the appendices but the structure chart representing their hierarchy is now going to be presented to give an idea of the complexity of the parsing tool. The structure chart of the parsing tool represented Figure 12 on page 49 contains some Unix scripts (xlyza, run_fd_parents, xparms, run_fd_children, run_parse_child_list, check_module_nature), some awk programs (lyza.awk, fd_parents.awk, lyzparm.awk, lyzout.awk, fd_children.awk, get_global_var_type.awk, fd_module_kind.awk), some sed programs (lyzparm.sed, join.sed, get_input.sed) as well as some C programs (del_line.c, count.c, append.c). The major program in the parsing tool is “lyza.awk”. Due to its importance, its algorithm is provided Figure 13 on page 50.
Figure 12. Structure chart for the parsing tool.
# lyza.awk algorithm

INPUT : None (lyza.awk will run on the call tree)
OUTPUT: file raw.data containing the information extracted by the parser

For each module definition line {
  - extract module name >> raw.data
  - determine its kind (global variable or function); kind >> raw.data
  - extract its type (int, float, char*, ...); type >> raw.data

  if ( module == global variable definition ) {
    - set to 0 the number of parameters; 0 >> raw.data
    - set to 0 the number of global variables accessed; 0 >> raw.data
    - set to 0 the number of function invoked; 0 >> raw.data
    - set to 0 the number of regular library modules invoked; 0 >> raw.data
    - set to 0 the number of io library modules invoked; 0 >> raw.data
  }

  if ( module == function ) {
    - extract parameters and their type from the source code
    - extract the list of the modules called by the current module
    - parse this list creating:
      - a list of the global variables accessed by the current module
        with their type
      - a list of the function invoked by the current module
      - a list of the regular library modules invoked by the current module
      - a list of the io library modules invoked by the current module
    - count the number of item in each list, store it at the first line
      - global variable list with their type >> raw.data
      - invoked function list >> raw.data
      - invoked regular library module list >> raw.data
      - invoked io library module list >> raw.data
  }

  - determine the list of the modules calling the current module
  - count the number of item in the list, store it at the first line
    - caller list >> raw.data
}

Figure 13. lyza.awk algorithm.
5.1.5 Output of the parsing tool

The flat file produced by the parsing tool contains all the information extracted from the call tree and the source code. Figure 14 on page 52 shows its format.
# Description of the format for the flat file raw data produced by xlyza and used to load the database.

<module name> <kind> <type>

--> The module can be a global variable or a function
It cannot be a library module because a library module is always invoked, not defined.
The kind can be 0 for a function
3 for a global variable
The type can be int, void, float, char *, etc....

--> A library module cannot be encountered at this level where the module name is actually defined. A library module,
by definition can only be invoked.
This justify the possible values of kind (0 or 3).
--> It must be noted that when xlyza recognises a library module, it checks the file system_io to see if the module
is defined as an io module.

<file name where is the module> <line number>
<number of parameters for the module at the function definition>
<param1> <type>
<param2> <type>
<....> <....>
<....> <....>
<....> <....>

<number of global variables accessed by the module>
<global var1> <type>
<global var2> <type>
<........> <type>
<........> <type>

<number of functions invoked by the module>
<invoked_fct1>
<invoked_fct2>
<........>
<........>

<number of regular library invoked by the module>
<invoked_lib1>
<invoked_lib2>
<........>
<........>

<number of io library invoked by the module>
<invoked_io1>
<invoked_io2>
<........>
<........>

<number of modules calling the module>
<caller1>
<caller2>
<....>
<....>

Figure 14. Format of the file raw data generated by the parser.
# In case there are no parameters at the function definition or in case no invoked module if the indicated kind are found, the number preceding the supposed invoked modules is set to zero. This flat file raw data will be used to load the database. It contains the extracted information from the ctree and the source code.

# The programs to extract these information use the following conventions:
- the kind is 0 for a function module invoked
- the kind is 1 for an io library module invoked
- the kind is 2 for a regular library module invoked
- the kind is 3 for a global variable accessed by the module

---

Figure 14. Continue.
5.1.6 Information storage

Characteristics of the chosen database

The database initially chosen to store the extracted information from the source code was the DBM database provided as a utility with the Unix operating system. Considering the relative simplicity of the accesses required by the ISCA, such a database fully satisfied the requirements. Steps were taken to use this database by Carol Lena Terry. To simplify this proof of concept study it was decided that a dynamically allocated data structure would be used instead. This choice prevents a multi-user utilization of the ISCA prototype and presents the disadvantage of requiring the destruction of the database at the exit of the application. However, the information is saved in the raw.data flat file and can be reloaded when necessary. Moreover a multi-user utilization does not represent a typical case of integration process. On the contrary, this choice presents the advantage of an optimized use of the memory on the computer where the application is running: the memory is dynamically allocated when required. It must be noticed that a database should be used in a production context to allow the multi-user utilization. This does not affect the functionality of the ISCA prototype in any way.

Definition of the data structure

The structure containing the information stored in the data structure is defined in the file data_struct.h shown in Figure 15 on page 55.
/*-----------------------------------------------*/
/* Thierry Le Gai  June 4, 1992 */
/* Header file containing the data structure */
/*-----------------------------------------------*/

#define IDENT_MAX 32
#define FILENAME_MAX 80

#define FTN 0 /* function kind */
#define IO 1 /* io library module kind */
#define LIB 2 /* regular library module kind */
#define VAR 3 /* global variable kind */

struct dbmdata_s {
  char fct_name[IDENT_MAX];
  char type[IDENT_MAX*4];
  int kind;
  char filename[FILENAME_MAX];
  int line_num;
  int parm_count; /* Number of parameters for the module */
  int logglobalvar_count; /* Number of global variable accessed by the function */
  int call_fct_count; /* Number of functions called by the module */
  int call_io_count; /* Number of io modules called by the module */
  int call_lib_count; /* Number of regular library modules called by the module */
  int calledby_count; /* Number of modules calling the module */
  struct parm_s *ptr_parm; /* Pointer to the parameters data structure */
  struct logglobalvar_s *ptr_logglobalvar; /* Pointer to the global variables data structure */
  struct call_fct_s *ptr_call_fct; /* Pointer to the called functions data structure */
  struct call_io_s *ptr_call_io; /* Pointer to the io modules data structure */
  struct call_lib_s *ptr_call_lib; /* Pointer to the library modules data structure */
  struct calledby_s *ptr_calledby; /* Pointer to the caller data structure */
  struct dbmdata_s *next;
};

struct parm_s {
  char name[IDENT_MAX];
  char type[IDENT_MAX*4];
  struct parm_s *next;
}; /* Data structure for the parameters of a function */

struct logglobalvar_s {
  char logglobalvar_fn[FILENAME_MAX];
  char type[IDENT_MAX];
}; /* Data structure for the global variables locally accessed by a module */

struct call_fct_s {
  char call_fct[FILENAME_MAX];
  struct call_fct_s *next;
}; /* Data structure for the functions called by a module */

struct call_io_s {
  char call_io[FILENAME_MAX];
  struct call_io_s *next;
}; /* Data structure for the io library modules called by a module */

struct call_lib_s {

Figure 15. Data structure.
char call_lib[Filename_MAX];
struct call_lib_s *next;
}; /* Data structure for the regular library modules called by a module */

struct calledby_s {
    char calledby_ftrn[Filename_MAX];
    struct calledby_s *next;
}; /* Data structure for the modules calling a module */
5.1.7 Extraction of information from the data structure

The program load_processes.c written in C language enables the extraction of information from the data structure and the display of the information on the ISCA user interface. Some of the functions in this program are called by the appropriate callback functions triggered by selecting the appropriate toggle buttons or the appropriate list items from the ISCA user interface. It must be noticed that the data structure is cleaned once the user exit the application. However, the file raw.data containing the data structure information is always available and can be reloaded.

5.1.8 Data structure updates

Once the data structure contains all the information of an application, the integrator can begin to design his integration functions. He will inquire the data structure during this task where he builds some data flow diagrams using Teamwork/SA. In case the analysis of the applications to be integrated was not led using a CASE system, the data flow diagrams are not available and the integrator will operate at the design level using Teamwork/SD. It is obvious that a perfect consistency between the actual code and the information contained in the data structure has to be kept. Consequently, if the integrator decides to work at a lower level during the integration task (modify or generate some code for example), he will be responsible for maintaining this consistency. He will then have to regenerate the new call tree for the updated application and he will need to reload the data structure. The integrator should generate the new code only when the design of all the integration functions has been done.

5.2 Application Interface

5.2.1 Prototype development using Motif

The user interface prototype has been developed using the Motif window manager. To reduce the time required to graphically define the widgets, Builder Xcessory from ICS Inc. has also been used. This interactive design tool for OSF/Motif enables a very easy widget geometry definition. Advantage was taken of the 3-D scultured look offered by Motif. The code which implements the user interface was connected with the code in-
quiring the database by using some callback procedures. The callbacks were defined and associated with the appropriate Motif primitives to generate the required actions. For example, the list containing the private modules of the currently analyzed application is loaded in the list container of the "Module list" widget when the toggle button "PRIV" of this widget is set to "on" by the integrator. A detailed description of the functions offered by the prototype is found in the following section.

5.2.2 ISCA prototype interface

The ISCA Display Manager has been designed to offer a very convivial interface to the integrator. MOTIF forms have been used to allow a dynamic resizing of the widgets proportionally to the resizing of each window. Figure 16 on page 59 presents a picture of the ISCA user interface. Four main windows can be described:

- The "Module List" window displays the list of the source code modules according to their type.
- The "Module detail" window displays the information associated with a module (name, type, filename, parameters with their type at the module definition, global variables accessed by the module, modules called by the module, modules calling the module.
- The "Source Code" window displays the filename of the source code containing the module currently selected in the "Module List" window, as well as the source code itself.
- The "Global Variable" window displays the list of the global variables contained in the source code or locally accessed by the module currently selected in the "Module List" window.

The interactivity of this interface needs to be pointed out. The integrator updates or displays the required information by selecting some toggle buttons or some push buttons with the mouse. He scrolls a list by selecting the scroll bar of the scrolled list. All those features greatly enhance the interactive consulting of the information stored in the data structure.
Figure 16. The ISCA interface.
5.2.3 Functions provided by the ISCA interface

This interface allows the user to inquire the data structure through an interactive user interface, namely the ISCA Display Manager developed using the Motif window manager. The user can:

1. Obtain a list of the modules for the application in the "Module List" widget
   - Choose to display either all the modules, only private ones, only library modules, or only Input/Output modules by selecting one of the toggle buttons "ALL", "PRIV", "LIB", or "I/O" with the mouse.
   - Find a particular module among this list by selecting the push button "FIND" and keying the module name in the displayed dialog widget popping up on top of the "Module List" widget as soon as the push button "FIND" is pushed.
   - Sort the displayed module list by alphabetic names
   - Select with the mouse a module of the list and update automatically all widgets in relationship with the information related to this module. For example, the source code corresponding to the current selected module in the "Module List" widget is automatically displayed in the "Source code" widget.

2. Visualize some information for this module in the "Module Detail" widget
   - Display the name of the selected module from the "Module list" widget
   - Display its type, location (file and line number), parameters (at the module definition)
   - Display a list of all the global variables accessed by the module
   - Display a list of the modules called by the selected module
   - Display a list of the modules calling the selected module

3. Get information regarding global in the "Global Variable List" widget
   - Display a list of all the global variables for the current analyzed application or display a list of the local variables that are locally accessed in the current selected module.
   - Display the type of a global variable selected in the global variable list as well as the modules where this global variable is accessed
6.0 Simulation of an integration

The description of the environment provided to the integrator will now be provided, followed by the description of the successive tasks through which the integrator will go through during an integration process of two pieces of software. Finally, a case study illustrating the "intellectual process" of integrating two very short programs will be presented. The use of the support provided by the Integration CASE workbench to achieve this integration will be described step by step.

6.1 Sketch of the environment

Figure 17 on page 63 depicts the functions provided to the integrator. The graphical interface of the ISCA is represented, providing the extracted information from the source code. The two Teamwork sessions running during an integration process are also represented. The "Integration session" enables the integrator to operate at the analysis level by designing or modifying some data flow diagrams of the software to be integrated. The "Consulting session" enables the integrator to visualize structure charts of the source codes to be integrated, giving precious help concerning the hierarchy of the module.

It must be noted that if the data flow diagrams of the software to be integrated are not available (the software may not have been analyzed using a CASE system before being designed), the integrator will operate at the desing level by creating or modifying some structure charts of the software to be integrated. The structure charts of the software to be integrated can always be obtained by running the Teamwork/C Rev reverse engineering tool on their source code.

The trimming processes enable the integrator to view part of a structure chart around a selected function. They are accessed from the "Consulting session" via some user menus.
developed using the Teamwork User Menu capabilities. The integrator chooses the number of levels above and below which the structure chart is trimmed around a selected function.

The import features enable the import of information in Teamwork calling a program developed using Teamwork/ACCESS programming language. They enable the integrator to attach a note to any kind of Teamwork object. Those features have been coded and integrated to the Integration CASE workbench. They can be invoked from a Teamwork user menu.
Figure 17. The integrator environment.
6.2 Course of action

The integrator's task is considered at the stage where the code concerned by the integration has been localized. It implies that some inquiries to the data structure have already been made using the ISCA user interface. Teamwork/C Rev has already been run on the source code of the applications to be integrated and a call tree has already been produced for each of them. The integrator wants to modify or create some data flow diagrams for the portion of the application he is interested in.

In parallel, the integrator has a Teamwork session running as well as the ISCA Display Manager. While modifying or creating a data flow diagram, the integrator can:

- Inquire the data structure concerning the application he is modifying to obtain any information he may need. This is done via the ISCA user interface. He can also browse the source code of the application if he needs it.
- Run some utilities (described below) to obtain some information concerning the actual hierarchy between the processes of the current application. Display the obtained structure charts via the Teamwork "Consulting session" running.

It must be noticed that the integrator still has to lead the "intelligent" part of his job by himself. Facilities to look for, display and make available any necessary information are provided as support, but the integrator is the only one leading the "intellectual process" of integrating the two software.

Once the integrator has created the desired data flow diagrams, they can be used to generate the corresponding structure charts. The module specifications can be loaded and new one can be created if necessary. The code for the new application can then be generated by running Teamwork C source builder.
6.3 Communication between the ISCA and Teamwork

Hierarchy

A visual help is provided to the integrator, which provides an image of the existing hierarchy between the functions he is interested in. This is achieved by providing several applications available from the Teamwork desktop (using Teamwork user-defined menus). The following four applications give great flexibility in visualizing the hierarchy between functions.

- **Create_SC_from_code** runs Teamwork/C Rev on a specified set of C source code modules to produce structure charts.
- **Create_SC_from_Ctree** runs Teamwork/C Rev on a call tree to produce structure charts.
- **build_ctree_fct_def** depicts the complete hierarchy of children below a given function. This application uses Unix script programs and the awk and sed parsing tools to create a localized call tree from which structure charts are automatically created using Teamwork/C Rev.
- **build_ctree_fct_level** creates structure charts for a set of modules surrounding a given function. A localized call tree is created and is used to automatically create some structure charts using Teamwork/C Rev. The user inputs the number of levels he wants to visualize above and below the function. Among the four developed applications currently presented, build_ctree_fct_level is the most complex. The structure chart describing its processes is provided in Figure 18 on page 66. This application uses Unix script programs (go, start_level, start_monte, monte, build_ctree_fct_level), some awk programs (get_line_number.awk, get_input.awk, fd_module_in_old_parent_list.awk, fd_parents.awk, fd_module.awk, get_module_line_number_in_ctree.awk, fd_children.awk, check_redun.dency.awk, fd_line_in_cree.awk, fd_children_level.awk), as well as some sed programs (update_child_list.sed, get_input.sed). It must be noticed that the complexity of build_ctree_fct_level is higher than the complexity of the parsing tool presented in Figure 12 on page 49. The two major processes composing this application are monte and build_ctree_fct_level. A description is provided for each of them in Figure 19 on page 67.
Figure 18. Structure chart for build_tree_for_level application.
# monte (Shell script 200 lines)
# Thierry Le Gal  July 2, 1992
#

INPUT :
- call tree to be trimmed
- function name around which the SC will be trimmed
- number of levels above the module to be trimmed

OUTPUT :
- line number in the call tree of the top module which will be represented on the SC

Note :
- choice input from the user when several possible parents for a module
- checking processes in case the top of the call tree is reached before the desired number of levels above the module to be trimmed

# build_ctree_fct_level (Shell script 190 lines)
# Thierry Le Gal  July 2, 1992
#

INPUT :
- call tree to be trimmed
- line number in the call tree where is the top module to be represented on the SC
- calculated number of levels below the top module to be represented

OUTPUT :
- the trimmed call tree

Note :
- necessity to check redundancy while creating the trimmed call tree (filtering process)

Figure 19. Description of the two major processes.
A performance evaluation of the slowest of these four applications, build_ctree_fct_level, has been done on the B-Spline module of ACSYNT, a design system written at Virginia Tech, which contains around thirty thousands lines of C source code. It appears that the number of lines in the call tree for the B-Spline module does not affect seriously the processing time. The number of lines in the call tree had to be increased by two thousands lines to double the processing time. The tests were run on an IBM RISC system 6000 and showed that this processing time was mainly due to the number of processes which are going to be displayed on the structure chart. For

- 249 children represented - the processing time was 20 minutes
- 94 children represented - the processing time was 6 minutes
- 15 children represented - the processing time was 1 minute

Considering the complexity of this application, the time response is considered as being reasonable. Proportionally, it takes about fifteen minutes to run Teamwork/C Rev on the call tree of the B-Spline module which contains thirty thousands lines.

It must be noted that the integrator will need to build only a few views of the process hierarchy of the software to be integrated.

**Loading a flat file in Teamwork**

The application "attach_note" has been coded using "C" and Teamwork/ACCESS. It enables the creation of a new note and the attachment of it to any of the following objects: data dictionary entry, data flow diagram, entity relationship diagram, process specification, matrix, structure chart, state transition diagram. The body of the note is loaded from a flat file. The functions are provided via a user-defined menu. This application improves the communication between Teamwork and the outside world by enabling the import of a flat file into the Teamwork environment. For example, it can be used to load some source code into a note.
6.4 Teamwork features

Teamwork provides the following features which will greatly facilitate the task of the integrator.

- **Buffer feature:** The same temporary buffer is used by Teamwork and the IBM RISC system 6000. The importance of this feature cannot be emphasized enough. For example, it enables the integrator to select information from the ISCA user interface, and copy it onto a data flow diagram.
  - A set of words highlighted with the mouse in an X-window environment can be pasted to any Teamwork object using the paste option of the Empty Space Menu.
  - A set of words in a Teamwork object can be highlighted with the mouse and pasted to the X-window environment by using the copy text options of the Empty Space Menu.
  - A set of words can be pasted from one Teamwork object to another Teamwork object by using the copy text or cut text options and the paste text option of the Empty Space Menu.

- **Copy note feature:** Any part of a note (title, body, part of the body) can be passed in the buffer and copied in any Teamwork object (buffer feature). A note attached to an object can be copied and associated with another Teamwork object by using the copy notes and paste notes options of the Empty Space Menu.

- **Show caller feature:** Show_callers is a Teamwork sample (the source code is not provided) which produces a report showing all of the objects within a Teamwork/SD model that call (invoke) a selected object in that model. From the report, a user can then highlight a sheet name or M-spec name and open it.

- **Autograf features:** Two of the Autograf features can be helpful to the integrator:
  - "Expand connector" merges the sheet named by a selected connector with the current sheet.
  - "Expand levels" recursively expands and merges with the current sheet a specified number of levels below the current sheet.

The functions developed by the author demonstrated the possibility of developing some communication mechanisms to access simultaneously Teamwork and the ISCA Display Manager.
6.5 Case study

The case study now presented aims to illustrate the use of the Integration CASE workbench environment while integrating two simple programs. It does not pretend to present an exhaustive use of the features provided by the environment. Only the use of some major features will be illustrated. The simplicity of the two applications to be integrated enables the reader to focus his attention on the use of the environment, instead of concentrating on the understanding of the two applications. The "intellectual process" of integrating the two simple programs is particular to the presented integration and should not be considered as being general in any case.

The two applications to be integrated had to satisfy the following characteristics:

- Written in C language
- Simple to be understood
- Necessitate the use of most of the features provided by the Integration CASE workbench to be integrated
- Illustrate the importance of global variables when integrating two applications

Using some of the features provided by the integration CASE workbench environment is not always necessary considering the simplicity of the two applications to be integrated. However, the integrator willingly uses those features to illustrate them. For example, the ISCA presents the advantage of helping the integrator to visualize the information extracted from the source code. This is particularly useful when the applications to be integrated contain thousands of lines of code. In the present case, the integrator uses the ISCA to visualize the information extracted from the source code although looking directly at the source code would be as efficient.

The "intellectual process" of integrating the two applications will now be explained step by step. The use of the environment will also be described.

6.5.1 Description of the two applications to be integrated

Most of the time, the source code of the two applications to be integrated is provided without any other information (no flow charts, no data flow diagrams and no structure charts). Therefore, the integrator will be placed in this situation in this case study. The
source code of the first application (appli1.c) is provided in Figure 20 on page 72. This very simple application displays on the screen the number of computers sold last year in the USA. This number is received as a command line argument from the user. Figure 21 on page 73 contains the source code of the second application to be integrated (appli2.c). appli2.c reads the average price of a computer from a flat file and displays it on the screen.

Those two separate applications are complementary. The integrator is willing to design a new application (appli12.c) sharing the data contained by appli1.c and appli2.c to display on the screen the amount of money earned by selling those computers. appli1.c and appli2.c are going to be embedded in appli12.c and invoked as subroutines. They will need to be slightly modified.
/* Thierry Le Gal */
/* appl1.c */
/* Purpose: illustrate the use of the Integration CASE workbench */
/* Note: To be considered as an example */
/* */
/* Appl1.c print the number of computers sold last year in the USA */
/* */
/* argv[1] contains the number of computers sold last year in the USA */

main(argc, argv)
int argc;  /* number of command line arguments */
char *argv[];  /* command line arguments */
{
    int num;

    sscanf(argv[1], "%d", &num);

    printf("The number entered was %dn", num);
}

Figure 20. appl1.c source code.
/* Thierry Le Gai */
/* appli2.c */
/* Purpose: illustrate the use of the integration CASE workbench */
/* Note: To be considered as an example */
/* */
/* Appli2.c read from a flat file the average price for a computer and display it on the screen */
/* */

#include <stdio.h>
#include <sys/file.h>

#define LINE_LEN 140
#define PRICE_FILE "price_file"

int average_price;

main()
{
  FILE *fp;
  char *line;

  line = (char *)malloc(LINE_LEN);
  if (line == (char *) NULL) {
    perror("not enough space on device, could not allocate memory");
  }

  /* Open the file containing the average price of a computer */
  if ((fp = fopen(PRICE_FILE,"r")) == NULL) {
    perror("error opening data file");
  }

  /* Get the line containing the average price for a computer */
  fgets(line,LINE_LEN,fp);
  fgets(line,LINE_LEN,fp);
  fclose(fp);

  /* Remove final line feed */
  line[strlen(line)-1] = '0';

  sscanf(line,"%d",average_price);

  printf("The average price of a computer is \%d\n",average_price);
}

Figure 21. appli2.c source code.
6.5.2 Description of the integration process

Step 1

The integrator needs to get a more in depth knowledge of the applications to be integrated. He needs to discover which functions will be concerned by the integration. He needs to know where the pertinent data in accomplishing this task is located. This is rarely easy and the ISCA can provide precious help to do so. The integrator runs Teamwork/C Rev on the two applications to be integrated. He obtains as output the files raw.data.appli1 and raw.data.appli2 presented in Figure 22 on page 77 and Figure 23 on page 78. Then he runs the parsing tool for each application to extract the information from the call tree and the source code. He visualizes the information via the ISCA Display Manager. He immediately realizes that the function “main()” in appli1.c is important for the integration as the local variable “num” contains the data he is interested in. For appli2.c, the function “main()” retains his attention as it contains the global variable “average_price” he is interested in. By localizing the functions accessing the data he was interested in, the integrator discovers the functions important for the integration.

Step 2

The integrator is willing to follow the data flows in the two applications he wants to integrate. He is going to lead the analysis of each application knowing that this step is not necessary but will help him to have a better understanding of what is going on. He opens a Teamwork/SA session and draws the data flow diagrams corresponding to appli1.c and appli2.c. During this process he uses the information displayed on the ISCA Display Manager, he scans the source code displayed in the “Source code” widget and selects with the mouse some data to be pasted on his data flow diagram’s sheets. The integrator comes out with several data flow diagrams or process specification presented in Figure 24 on page 79, Figure 25 on page 80, Figure 26 on page 81 for appli1.c and Figure 27 on page 82 and Figure 28 on page 83 for appli2.c.
Step 3

Taking advantage of the previous step, the integrator analyzes what is going to be appli12.c. He opens a new model on Teamwork/SA and draws the context diagram presented in Figure 29 on page 84 where "calculate_computer_sales" is the main process and where "user" and "screen" are the two terminators. He draws the data flow diagram "0" describing the process "calculate_computer_sales". This data flow diagram presented in Figure 30 on page 85 contains the processes "get_num_computer_sold" and "get_average_computer_price" defined in the models for appli1.c and appli2.c. He uses the "copy subtree" and "paste text" features of the Empty Space Menu to bring these sheets into the current model. At this point, the analysis of the integrated application is finished. From there, the integrator can derive his data flow diagrams to obtain the corresponding structure charts. This is not what he chooses to do. He decides to operate at the design level by designing the integrated application himself after having analyzed it.

Step 4

The integrator opens the structure charts "main" obtained for appli1.c and appli2.c after having run Teamwork/C Rev in step 1. He builds the structure chart "main" presented in Figure 31 on page 86, for the new appli12.c integrated application. The two main functions of appli1.c and appli2.c are invoked by the "main" process of appli12.c under the name "get_num_computer_from_user" and "get_average_price_from_read_file".

Step 5

The integrator attaches a module specification to each of those two processes and loads them respectively with the source code of appli1.c and appli2.c. This is done by using the "paste from file" option of the Empty Space Menu. The filename of appli1.c and appli2.c are respectively indicated (path included) by the integrator.

Step 6

The integrator runs Teamwork/C Source Builder to generate the source code appli12.c of the integration (refer to (Team10)).
Step 7

The integrator checks the obtained source code and creates the executable. Depending on the level of experience of the integrator, some modifications at the design or at the source code level may be necessary. In the present case, the integrator forgot to remove the "printf" process below the "get_num_computer_from_user" and "get_average_price_from_file" processes. As only the "sale" data was required to be displayed by appli12.c, those "printf" processes have to be removed. It can be done at the design level or at the source code level. The integrator choses to comment those functions at the source code level, knowing that he will regenerate a new structure chart from the source code appli12.c running Teamwork/C Rev to insure the consistency between the source code and the designed structure chart. It must be noticed that the integrator defined the two global variable "glo_average_price" and "glob_num_computer_sold" on the structure chart (Figure 31 on page 86) in order to achieve the integration. Those global variables enable him to access the local variable "num" and the local variable "average_price" in the invoked appli1.c and appli2.c applications. The integrator also modified the name of the two invoked function to "get_num_computer_from_user" and "get_average_price_from_file". The two define statements of the invoked appli1.c had to be moved to the top of appli12.c to avoid a compiler error.

Step 8

The integrator runs Teamwork/C Rev to regenerate the structure chart corresponding to the source code appli12.c and insure the consistency between the source code and the designed structure chart. The integrated application’s source code is presented in Figure 32 on page 87.
main 0 int
/u/legal/ibm/parser/these/appli1.c 12
2
argc[] char *
argc int
0
0
1
sscanf
1
printf
0

Figure 22. raw.data.appli1 flat file.

Simulation of an integration
average_price 3 int
/u/legal/ibm/parser/these/appli2.c 16
0
0
0
0
0
1
main
main 0 int
/u/legal/ibm/parser/these/appli2.c 18
0
1
average_price int
0
7
malloc
printf
open
fopen
fgetc
fclose
strlen
sscanf
1
printf
0

Figure 23. raw.data.appli2 flat file.
Figure 24. appli.c Context-Diagram.
Figure 25. appl1.c data flow diagram 0.
NAME:
1:2

TITLE:
convert_format

INPUT/OUTPUT:
input_from_keyboard : data_in
output_to_screen : data_out

BODY:
Process to copy the input_from_keyboard data into the
output_to_screen data by taking care of the different
format between these two data flows.
Use the function 'scanf'

Figure 26. appl1.c process specification 1.
Figure 27. appl2.c Context-Diagram.
Figure 28. appl2.c data flow diagram 0.
Figure 29. appl12.c Context-Diagram.

Simulation of an integration
Figure 30. appl12.c data flow diagram 0.
/* Thierry Le Gal */
/* appli12.c */
/* Purpose: illustrate the use of the Integration CASE workbench */
/* Note: To be considered as an example */
/* appli12.c is the result of the integration of appli1.c and appli2.c */

#include <stdio.h>
#include <sys/file.h>

/* Function definition */
void get_num_computer_from_user();
void get_average_price_from_file();

/* Global variables */
int glob_num_computer_sold;
int glob_average_price;

main()
{
    int sales;

    get_num_computer_from_user();
    get_average_price_from_file();

    sales = glob_num_computer_sold * glob_average_price;
    printf("Sales of computers last year: $ %dn\n", sales);
    printf("End\n");
}

/* Thierry Le Gal */
/* appli1.c */
/* Purpose: illustrate the use of the Integration CASE workbench */
/* Note: To be considered as an example */
/* Appli1.c print the number of computers sold last year in the USA */

/* argv[1] contains the number of computers sold last year in the USA */

void get_num_computer_from_user(argc, argv)
int argc;     /* number of command line arguments */
char *argv[]; /* command line arguments */
{
    int num;

    sscanf(argv[1], "%d", &num);

    glob_num_computer_sold = num;

    printf("The number of computers sold last year in the USA is %dn\n", num);
}

Figure 32. appli12.c source code.
/* Thierry Le Gal */
/* appl2.c */
/* Purpose: illustrate the use of the integration CASE workbench */
/* Note: To be considered as an example */
/* */
/* Appl2.c read from a flat file the average price for a computer and display it on the screen */
/* */
/* ---------------------------------------------------------------------------------------------------------------------------------- */

#define LINE_LEN 140
#define PRICE_FILE "price_file"

int average_price;

void get_average_price_from_flat_file()
{
    FILE *fp;
    char *line;

    line = (char *)malloc(LINE_LEN);
    if (line == (char *)NULL) {
        perror("not enough space on device; could not allocate memory");
    }

    /* Open the file containing the average price of a computer */
    if ((fp = fopen(PRICE_FILE, "r")) == NULL) {
        perror("error opening data file");
    }

    /* Get the line containing the average price for a computer */
    fgets(line, LINE_LEN, fp);
    fgets(line, LINE_LEN, fp);
    fclose(fp);

    /* Remove final line feed */
    line[strlen(line)-1] = '\0';

    sscanf(line, "%d", &average_price);

    glob_average_price = average_price;

    /* printf("The average price of a computer is %dn",average_price); */
}

Figure 32. Continue.
7.0 Conclusions

The goals of this research have been achieved. The ISCA has been developed to enable the storage of relevant information in a database and make them available to the integrator via the interactive ISCA user interface. Some Teamwork features providing help to the integrator have been pointed out. Some applications to improve the assistance offered to the integrator as well as the communications between Teamwork and the ISCA have been developed.

The ISCA in itself appeared as a very valuable tool to lead the analysis of source code as well as to lead re-engineering operations. The potential gains of productivity obtained by using the ISCA in a re-engineering operation are expected to be significant. The improvements offered by the ISCA are expected to be as important as those due to the release of the first generation of interactive debuggers.

Some recommendations will now be presented.

The first recommendation concerns the nature of the source code to be analyzed. The parsing tool is running on software written in C language. However, much engineering software is written in FORTRAN, that is why the parsing tool should also run on FORTRAN code too. This does not present any major difficulty.

Another recommendation concerns the testing and the improvement of the Integration CASE workbench environment. Before commercialization of this product, the comments concerning this environment with its tools and features should be collected from professional integrators. They should lead some integration operations using the ISCA for several months to improve this product so that it fully satisfies their needs. The importance of this phase shouldn’t be underestimated as well as the amount of time it will require. If the ideas developed in the Integration CASE workbench are as valuable as it
is anticipated, a new investigation should be led in a common effort with some experienced integrators.

The ISCA in itself is a subsequent product which can be commercialized independently of the environment provided. Knowing the budget envelopes allocated to maintenance operations, there is no doubt that companies are waiting for such a tool to improve their maintenance efficiency and realize drastic savings on their maintenance budget. The Integration CASE environment should be considered as a proof of concept prototype which needs to be reanalyzed by a team of experts in maintenance tasks before being redeveloped and commercialized on a large scale.
8.0 References

(Brit90) R. N. Britcher, IBM Systems Journal, 1990, vol29 n4


(Knol91) Edward L. Knoll, Teamworkers Newsletter, fall 91, pp. 13-14


(Team1) CADRE, "Teamwork/SD; User's Guide", Release 4.0.1

(Team2) CADRE, "Teamwork; Environment Reference Manual"; Release 4.0.1

(Team3) CADRE. "Teamwork/SA, Teamwork/RT; User's Guide", Release 4.0.1

(Team4) CADRE, "Teamwork; System Administrator Manual", Release 4.0.1

(Team5) CADRE, "Teamwork/IM; User's Guide", Release 4.0.1

(Team6) CADRE, "Teamwork/C Rev; User's Guide", Release 4.0.1

(Team7) CADRE, "Teamwork/FORTRAN Rev; User's Guide", Release 4.0.1

(Team8) CADRE, "Teamwork/ACCESS; User's Guide", Release 4.0.1

(Team9) CADRE, "Teamwork Toolkit Utilities; User's Guide", Release 4.0.1

(Team10) CADRE, "Teamwork C Source Builder; User's Guide", Release 4.0.1

(Team11) CADRE, "Teamwork/User Menus; User's Guide", Release 4.0.1

References


(VPI90) "A Research Report to IBM Corporation", Virginia Polytechnic Institute & State University, USA, December 1990.


Appendix A

This appendix presents the programs used to trim a call tree around a given process. A description of the purpose of each function, the inputs and the outputs is given at the beginning of each program.

Files presented:

- start.monte
- start.def
- build_ctree_fct_def
- update_child_list.sed
- check_redundency.awk
- get_input.sed
- fd_children.awk
- fd_module.awk
- get_module_line_number_in_ctree.awk
- fd_parents.awk
- count_lines.awk
- get_input.awk
- fd_module_in_old_parent_list.awk
- get_line_number.awk
- build_ctree_fct_level
- monte
- fd_line_in_ctree.awk
- fd_children_level.awk
- start.level
- go
# Thierry Le Gal       April 12, 1992
# start.monte
#

# Shell script running the shell script monte. Returns the line number of the desired parent in the call tree
# is invoked by start.level
# $1 is the call tree (with its path) that is going to be used to find the parents
# $2 is the function name for which we are looking at the parents
# $3 is the number of levels above the fct we want to go
#tree=/u/legal/twk/calltree/ctree/tree1.o
#function=get_input
#number_of_levels_above=3

tree=$1
function=$2
number_of_levels_above=$3
line_number=-1

echo Search for parents ...
while (test $line_number -eq -1)
do
    # search the parent tree and store the line number of the found parent in result file
    /u/legal/twk/calltree/awk/monte $tree $function $number_of_levels_above

    # read the line number of the highest parent from result file
    line_number=`awk -f get_line_number.awk result`
    rm result

    if test $line_number -eq -1
    then
        echo Previous number of levels: $number_of_levels_above
        echo Enter the new number of levels to search for above the function $function
        read number_of_levels_above
    fi

done
# store line_number in the file result1
    echo $line_number > result1
echo Search for parents ends successfully !!!
# Shell script to trim the calltree for a function, give the hierarchy below it and generate some structure charts

# $1 is the name (with its path) of the call tree to be trimmed
# $2 is the name of the function
# $3 is the name of the model where to put the structures charts
# $4 is the name of the config file to be used
# $5 is the title for the structure charts

# Trim the calltree to obtain the hierarchy below a function
/u/central/twk/calltree/awk/build_ctree_fct_def $1 $2

cree_name=/u/central/twk/calltree/cTREE/$2.def

# Run T/Crev on this ctree

echo Run T/Crev on the ctree $cree_name
echo Using the config file $4
echo Store the SCs in the model $3
echo T/Crev running ...
/cadre/tool_kit/crev -config $4 -model $3 -title $5 -call_tree $cree_name
echo End successfully !!!
#--
# Thierry Le Gal  April 15, 1992
# build_cTREE.fct_def
#

# Shell script to build the call tree below a given function
# is invoked by start.def
# $1 is the name (with the path) of the call tree to be trimmed
# $2 is the name of the function for which we want to build the call tree

module=$2
echo Creating the call tree below the function $2
echo Trim the call tree $1
echo Trimming ...

go into the right directory
cd /u/legal/twk/calltree/awk

# set the value of FUNCTION_NAME in fd_children.awk
sed 's/FUNCTION_NAME/$module/g' fd_children.awk > tmp_fd_children.awk

# set the value of object_file
object_file=/u/legal/twk/calltree/cTREE/$module.def

# flush $object_file before beginning
echo " " > $object_file

# rm the first line in $object_file
sed -f update_child_list.sed $object_file > tmp
cat tmp > $object_file
rm tmp

# flush child_list file before beginning
echo " " > child_list

# rm the first line in child_list
sed -f update_child_list.sed child_list > tmp
cat tmp > child_list
rm tmp

# loop till the child_list is empty
while test $module
  do
    # set redundancy to 1 to enter the loop the first time
    redundancy=1

    # read child_list till a new module is found
    while test $redundancy -eq 1
      do
        # set the value of scanned_module in check_redundancy.awk
        sed 's/scanned_module/$module/g' check_redundancy.awk > tmp_check_redundancy.awk

        # scan the object_file (new fct tree) to check the existence of the definition for $module
        awk -f tmp_check_redundancy.awk $object_file $redundent_module=\'sed -n -f get_input.sed redundent_file'\n        rm redundent_file
    
    # read the value returned by tmp_check_redundancy.awk
    #...
if test "$module" != "$redundent_module"
  then
    redundancy=0
  else
    # get the next module in child_list
    module=`sed -n -f get_input.sed child_list`

    # rm the first module in child_list
    sed -f update_child_list.sed child_list > tmp
    cat tmp > child_list
    rm tmp

  fi

done

# set the value of parent_name in fd_children.awk
sed '6 s/parent_name/"$module"/' tmp_fd_children.awk > tmp1_fd_children.awk

# get the children for the module : append them to child_list, update the object_file (new fct tree)
awk -f tmp1_fd_children.awk $1
rm tmp1_fd_children.awk

# get the next module in child_list
module=`sed -n -f get_input.sed child_list`

# rm the first module in child_list
sed -f update_child_list.sed child_list > tmp
cat tmp > child_list
rm tmp

done

rm child_list
rm tmp_fd_children.awk
rm tmp_check_redundency.awk
echo End successfully !!!
# Delete the first line of a file; copy all the others as they were

```plaintext
1 {d
}
2.{h
}
```
# AWK file to scan the file find if scanned_module is defined in it as a caller
# scanned_module must be set at the fct call
# return the scanned_module value in case it was found as a caller
# Is invoked by build_cintree_fct_level

{
    scan_file_for("scanned_module");
}

# create the file redundant_file in case no redundancy was found

END {
    printf("\n\n\n") >> "/u/legal/twk/calltree/awk/redundent_file";
}

function scan_file_for(MODULE) {
    module_name = substr($2,1,length($2)-1);
    if ($1 == 1) {
        current_caller = module_name;
        if ( current_caller == MODULE ) {
            printf("\n\n\n") >> "/u/legal/twk/calltree/awk/redundent_file";
        }
    }
}

}
# Get the first line of a file

```
1 {h
p
q
}
```
# AWK file to find the definition for a given FUNCTION_NAME. Get the modules immediately called by this module
# parent_name must be set at the fct call
# is invoked by build_cmtree_fct_level

{
    get_children("parent_name");
}

function get_children(MODULE)
{
    module_name = substr($2,1,length($2)-1);
    if ($1 == 1) {
        current_caller = module_name;
        if (current_caller == MODULE) {
            caller_name = current_caller;
            caller_line_number = FNR;
            caller_level = $1;
            printf("%s\n", $0) >> "/u/legal/twk/calltree/cTREE/cTREE_FUNCTION_NAME.def";
        }
    }
    if ($1 == caller_level+1 && (current_caller == MODULE)) {
        called_name = substr($2,1,length($2)-1);
        called_line_num = FNR;
        called_level = $1;
        printf("%s\n", $0) >> "/u/legal/twk/calltree/cTREE/cTREE_FUNCTION_NAME.def";
        printf("%s\n", module_name) >> "/u/legal/twk/calltree/awk/child_list";
    }
}
# Thierry Le Gal          April 12, 1992
# ld_module.awk
#
# AWK file to get the module itself in the calltree
# child_name must be set at the fct call
# is invoked by monte
#
{
    get_parents("child_name");
}

function get_parents(MODULE)
{
    module_name = substr($2,1,length($2)-1);
    if (( $1 == 1) && (module_name == MODULE) ) {
        parent = module_name;
        parent_line_number = FNR;
        printf("%d, %s\n", parent_line_number, parent) >> "/uflegal/twk/calltree/awk/parent_list";
    }
}
# AWK file to get the first field of the first line of a file
# The format of the file is: number, <space> module_name
# Is invoked by monte

{
    if (FNR == 1)
    {
        line_number = substr($1, 1, length($1) - 1);
        print line_number;
    }
}

# Thierry Le Gal  
# April 12, 1992  
# fd_parents.awk  
#

# AWK file to get the modules immediately calling the considered child  
# child must be set at the tct call  
# is invoked by monte

{
    get_parents("child_name");
}

function get_parents(MODULE)
{
    module_name = substr($2,1,length($2)-1);
    if ($1 == 1) {
        potential_parent = module_name;
        potential_parent_line_number = FNR;
    }
    if ($1 == 2) && (module_name == MODULE) {
        parent = potential_parent;
        parent_line_number = potential_parent_line_number;
        print("%d, %s\n", parent_line_number, parent) >> "/u/legal/twk/calltree/awk/parent_list";
    }
}
# count_lines.awk

BEGIN { number_of_lines = 0 }
{
   if ($1 != "") number_of_lines = number_of_lines + 1
}
END { printf number_of_lines }
# Thierry Le Gal        April 12, 1992
# get_input.awk
#

# AWK file to get the module in the first line of a file
# The format of the file is: number, <space> module_name
# Is invoked by monte

{ if ( FNR == 1 )
  module_name = substr($2,1,length($2));
  print module_name;
}
# AWK file to get the line of the file containing MODULE
# The format of the file is: number, <space> module_name
# Is invoked by monte

if ( $2 == "MODULE" )
{
  line = $0;
  print line;
}

# Thierry Le Gal April 12, 1992
# get_line_number.awk
#-----------------------------------------------

# AWK file to get the line number stored in the first line of a file. Format of the file is: <number>
# The format of the file is: number

{
if ( FNR == 1 )
{
    line_number = substr($1,1,length($1));
    print line_number;
}
}
# Thierry Le Gal        April 12, 1992
# build_cTREE.fct_level
#
# Shell script to build the call tree for the call of a function
# $1 is the name (with the path) of the call tree to be trimmed
# $2 is the line number in the call tree where is the considered function
# $3 is the number of levels below the function call we want to represent in the SC

old_level=0
count=0

# go into the right directory
cd /u/legal/twk/calltree/awk

# set the value of LINE_NUMBER in fd_line_in_cTREE.awk
   sed 's/LINE_NUMBER/$2/g' fd_line_in_cTREE.awk > tmp_fd_line_in_cTREE.awk

# find the module associated with the line number of the call tree
   module=`awk -f tmp_fd_line_in_cTREE.awk $1`
   rm tmp_fd_line_in_cTREE.awk

# prompt the user
   echo Creating a call tree $3 levels below the function $module defined line $2
   echo of the call tree $1
   echo Trimming ...

# set the value of FUNCTION_NAME in fd_children.awk
   sed 's/FUNCTION_NAME/$module/g' fd_children_level.awk > tmp_fd_children.awk

# set the value of the object_file (contains the new call tree generated)
   object_file=/u/legal/twk/calltree/cTREE_$module.level

# flush $object_file before beginning
   echo " " > $object_file

# rm the first line in $object_file
   sed -f update_child_list.sed $object_file > tmp
   cat tmp > $object_file
   rm tmp

# flush child_list before beginning
   echo " " > child_list

# rm the first line in child_list
   sed -f update_child_list.sed child_list > tmp
   cat tmp > child_list
   rm tmp

# initialize the value of written_level and read_level
   written_level=1
   read_level=1

# loop till the child_list is empty or the number of levels has been covered
while test $module
   do
# exit when the number of levels is reached
   if test $written_level -gt $3
     then
       rm child_list
       rm tmp_fd_children.awk
       rm tmp1_fd_children.awk
     fi
   done
rm tmp_check_redundency awk
echo number of child scanned $count
  echo End successfully !!!
  exit
fi

# set redundancy to 1 to enter the loop the first time
redundancy=1

# read child_list till a new module is found
while test $redundency -eq 1
do

# set the value of scanned_module in check_redundency.awk
  sed 's/\$scanned_module/\$module/g' check_redundency.awk > tmp_check_redundency.awk

# scan the object file (new fct tree) to check the existence of the definition for $module
  awk -f tmp_check_redundency.awk $object_file

# read the value returned by tmp_check_redundency.awk
  redundant_module=`sed -n -f get_input.sed redundant_file`
  rm redundant_file

  if test "\$module" != "$redundent_module"
    then
      redundancy=0
    else
      # get the next line in child_list (contains level + module name)
      next_line=`sed -n -f get_input.sed child_list`

      # set the shift_flag to extract the module from next_line variable
      read_level=`echo $next_line | cut -c1-2`
      if test $read_level -le 9
        then
          shift_flag=1
        else
          shift_flag=2
        fi
      fi

      # get the next_module extracted from next_line variable
      if test $shift_flag = 1
        then
          read_level=`echo $next_line | cut -c1-1`
          module=`echo $next_line | cut -c3-75`
        else
          read_level=`echo $next_line | cut -c1-2`
          module=`echo $next_line | cut -c4-75`
        fi

      rm the first module in child_list
      sed -f update_child_list.sed child_list > tmp
      cat tmp > child_list
      rm tmp
    fi

done

# set the value of LEVEL in fd_children_level.awk
  sed 's/\$written_level/\$level/g' tmp_fd_children.awk > tmp1_fd_children.awk

# set the value of parent_name in fd_children.awk
  sed 's/\$parent_name/\$module/g' tmp1_fd_children.awk > tmp2_fd_children.awk
```bash
# get the children for the module: append them to child_list, update the object_file (new fct tree)
awk -f tmp2_fd_children.awk $1
rm tmp2_fd_children.awk

# get the next line in child_list (contains level + module name)
next_line=`sed -n '/get_input.sed child_list'` echo next line in child_list $next_line
count=`expr $count + 1`

# exit in case $next_line is empty
# this is not necessary but avoid an exit error when reaching if test $read_level -le 9 due
# to $read_level = " " in that case
if test -z "$next_line"
then
  rm child_list
  rm tmp_fd_children.awk
  rm tmp1_fd_children.awk
  rm tmp_check_redundancy.awk
  echo number of child scanned $count
  echo End successfully !!!
  exit
fi

# set the shift_flag to extract the module from next_line variable
read_level=`echo $next_line | cut -c1-2`
if test $old_level -ne $read_level
then
  echo level $read_level
  old_level=$read_level
fi
if test $read_level -le 9
then
  shift_flag=1
else
  shift_flag=2
fi

# get the next_module extracted from next_line variable
if test $shift_flag = 1
then
  read_level=`echo $next_line | cut -c1-1`
  module=`echo $next_line | cut -c3-75`
else
  read_level=`echo $next_line | cut -c1-2`
  module=`echo $next_line | cut -c4-75`
fi

# increment the value of written_level
written_level=`expr $read_level + 1`

# rm the first line in child_list
sed -i update_child_list sed child_list > tmp
cat tmp > child_list
rm tmp

done
rm child_list
rm tmp_fd_children.awk
rm tmp1_fd_children.awk
rm tmp_check_redundancy.awk
echo number of child scanned $count
echo End successfully !!!
```
# Shell script to enable you to define uniquely the parents' process of a process in a call tree
# The output is the line number of one of the parents process. You go up in the hierarchy till one of the selected
# is unique or till you decide to enter its line number in the call tree (which defines it uniquely)
# is invoked by start.monte

# $1 is the name (with the path) of the call tree to be trimmed
# $2 is the name of the function around which we want to represent the SC
# $3 is the number of levels desired above the function $2

current_child=$2
desired_number_level_above=$3
current_level_above=0
next_parent="bidon"
first_time=0
old_parent_list_existence=0

# go into the right directory
cd /u/legal/tk/calltree/awk

if test $desired_number_level_above -eq 0
then
  # flush parent_list file
  echo " " > parent_list

  # rm the first line in parent_list
  sed -f update_child_list sed parent_list > tmp
  cat tmp > parent_list
  rm tmp

  # set the value of child_name in fd_parents.awk
  sed 's/child_name/"$current_child"/f' fd_module.awk > tmp_fd_module.awk

  # get the parents for the module: store them in parent_list
  awk -f tmp_fd_module.awk $1
  rm tmp_fd_module.awk
  line_number="awk -f get_module_line_number_in_ctrie.awk parent_list"

  # store the line_number in result file
  echo "$line_number" > result

  echo Reached $current_level_above level above the function $2;

  # save the actual number of levels above the function;
  echo $current_level_above > level_above_fct;
  echo Current module:
  cat parent_list

  rm parent_list
  exit
else
fi

# loop until a unique parent is found or the number of levels above the fct is reached
while ( (test $current_level_above -lt $desired_number_level_above) && (test "$next_parent" != "") )
do

  if test $first_time -eq 1
  then
  # save parent_list file in case the next parent_list would be empty (reaching the top of a branch)
  fi

  # do the following operations:

  # get the parents for the module: store them in parent_list
  awk -f tmp_fd_module.awk $1
  rm tmp_fd_module.awk

  line_number="awk -f get_module_line_number_in_ctrie.awk parent_list"

  # store the line_number in result file
  echo "$line_number" > result

  echo Reached $current_level_above level above the function $2;

  # save the actual number of levels above the function;
  echo $current_level_above > level_above_fct;
  echo Current module:
  cat parent_list

  rm parent_list
  exit
  fi

113
cat parent_list > old_parent_list
old_parent_list_existence=1
fi
# set first_time to 1 after we passed the previous test for the first time
first_time=1

# flush parent_list file
    echo " " > parent_list

# rm the first line in parent_list
    sed -f update_child_list sed parent_list > tmp
cat tmp > parent_list
rm tmp

# set the value of child_name in fd_parents.awk
    sed 's/child_name/$current_child/' fd_parents.awk > tmp_fd_parents.awk

# get the parents for the module : store them in parent_list
    awk -f tmp_fd_parents.awk $1
    rm tmp_fd_parents.awk

# echo parent_list should not appear
# cat parent_list

# set the value of multiple_parents_for_current_module
# 1 if a single parent
# 2 or more if multiple parents (depending on the number of parents)
    multiple_parents_for_current_module=`awk -f count_lines.awk parent_list`

# save the value of multiple_parents_for_current_module
    multiple_parents=$multiple_parents_for_current_module

# set the value of next_parent if the desired level is not reached
if test 'expr $current_level_above + 1' -lt $desired_number_level_above
then
    if test $multiple_parents_for_current_module -ge 2
    then
        echo Parent_list:
cat parent_list
echo Multiple parents, enter your choice:
read next_parent
    fi
    # to save the value of current_child in case next_parent ="
    if test "$next_parent" = ""
    then
        current_child=$next_parent
current_level_above=`expr $current_level_above + 1`
    fi
    # reset multiple parents to 1 (a single parent)
    multiple_parents_for_current_module=1
else
    # read the next_parent value from the parent_list
    awk -f get_input.awk parent_list
    next_parent=
    # to save the value of current_child in case next_parent ="
    if test "$next_parent" = ""
    then
        current_child=$next_parent
current_level_above=`expr $current_level_above + 1`
    fi
fi
is going to leave the while loop because of reaching the desired level
# echo you are going to exit the while because you reach the level `expr $current_level_above + 1`
# check that parent_list is not empty at the same time
next_parent= awk -f get_input.awk parent_list
# next_parent is set to "" in case parent_list is empty, allow us to exit the while loop
# then we don't need to increment current_level_above to exit the while loop
# if test "$next_parent" != ""
then
# increase the current_level_above to exit the while loop
current_level_above= expr $current_level_above + 1'
fi

# check if the parent_list is empty or not
# if parent_list empty set next_parent = ""
# else increase the value current_level to exit the while
fi
done

# we exited the while loop before reaching the desired number of levels
if test "$next_parent" = ""
then
  echo Reached the top of a branch
  echo Only `expr $current_level_above` levels above the function $2 will be represented
# save the actual number of levels above the function
  echo $current_level_above > level_above_fct

# set the value of MODULE in fd_module_in_old_parent_list.awk
  sed 's/MODULE/$current_child/g' fd_module_in_old_parent_list.awk > tmp_fd_module_in_old_parent_list.awk

# get the line associated with $current_module_in_old_parent_list
  line= awk -f tmp_fd_module_in_old_parent_list.awk old_parent_list
  rm tmp_fd_module_in_old_parent_list.awk

echo Current and top module:
  echo $line
  line_number= awk -f get_module_line_number.in_ctree.awk old_parent_list;
else
  # we exited the while loop by reaching the desired number of levels
  multiplicity= awk -f count_lines.awk parent_list

  case $multiplicity in
  # case 0 will never occur because next_parent will be ""
  # 0) echo you reached $current_level_above levels above the function $2;
  # echo next_parent $next_parent;
  # echo $current_child;
  1) echo Reached $current_level_above levels above the function $2;
  # save the actual number of levels above the function;
  # echo $current_level_above > level_above_fct;
  # echo Current module:
  # cat parent_list;
  line_number= awk -f get_module_line_number.in_ctree.awk parent_list ;;
  *) echo Reached $current_level_above levels above the function $2;
  # save the actual number of levels above the function;
  # echo $current_level_above > level_above_fct;
  # echo Previous module had the multiple parents;;
  # cat parent_list;
  echo "Define a unique parent:";
  echo " - enter -1 to increase the number of levels in the search";
  echo " - enter a call tree line number to determine a unique parent";
  read line_number ;;
  esac
fi

# store the line_number in result file
  echo $line_number > result

rm parent_list
if test $old_parent_list_existence -eq 1
  then
    rm old_parent_list
fi
# Thierry Le Gal        April 12, 1992
# fd_line_in_cctree.awk
#

# AWK file to find the module name at a line number in a call tree
# Is invoked by build_cctree_cct_level

{ 
  if ( FNR == LINE_NUMBER ) 
  { 
    module_name = substr($2,1,length($2)-1); 
    print module_name; 
  } 
}
# Thierry Le Gal            April 12, 1992
# ld_children_level.awk
#

# AWK file to find the definition for a given FUNCTION_NAME
# Get the modules immediately called by a given module (parent_name)
# parent_name must be set at the fct call
# is invoked by build_cTREE_fct_level

{
    get_children("parent_name");
}

function get_children(MODULE)
{
    module_name = substr($2,1,length($2)-1);
    if ($1 == 1) {
        current_caller = module_name;
        if (current_caller == MODULE) {
            caller_name = current_caller;
            caller_line_number = FNR;
            caller_level = $1;
            printf("%s\n", $0) >> "/u/legal/twk/calltree/cTREE/cTREE_FUNCTION_NAME.level";
        }
    }
    if ($1 == caller_level+1) && (current_caller == MODULE) {
        called_name = substr($2,1,length($2)-1);
        called_line_num = FNR;
        called_level = $1;
        printf("%s\n", $0) >> "/u/legal/twk/calltree/cTREE/cTREE_FUNCTION_NAME.level";
        printf("%s %s\n", LEVEL, module_name) >> "/u/legal/twk/calltree/awk/child_list";
    }
}

# Shell script to trim the calltree around a given function and generate some structure charts
# is invoked by go
# $1 is the name (with its path) of the call tree to be trimmed
# $2 is the name of the concerned function
# $3 is the number of levels above the function call we want to represent in the SC
# $4 is the number of levels below the function call we want to represent in the SC
# $5 is the name of the model where to store the structure charts
# $6 is the name of the config file to be used
# $7 is the title for the structure charts

# find the line number in the call tree of the fct at the desired level above the fct; store it in the file result1
/u/legal/twk/calltree/awk/start.monte $1 $2 $3

# read the actual number of levels obtained
# $3 is not necessarily the right number of levels above the fct, monte may have changed it
# ex: extend the research to more levels above the fct
#    reach the top of a branch
#    number_levels_above=`awk -f get_line_number.awk level_above_fct`
#     rm level_above_fct

# read the line number of the highest parent from result file
line_number=`awk -f get_line_number.awk result1`
rm result1

# trim the calltree to obtain the hierarchy around the function
/u/legal/twk/calltree/awk/build_cTREE_fct_level $1 $line_number `expr $number_levels_above + $4`

# is not right because $3 may have been changed in build_cTREE_fct_level
# ex: extend the research to more levels above the fct
#    reach the top of a branch

# set the name of the obtained ctree on which to run T/Crev
ctree_name=`u/legal/twk/calltree/ctree/ctree_$2.level`

# Run T/Crev on this ctree
#---------------------------------------------------------
# Syntax /coder/tool_kit/crev [-config config_file] [-model model_name] [-title string] [-call_tree call_file]
#---------------------------------------------------------
#echo Run T/Crev on the ctree $ctree_name
#echo Using the config file $6
#echo Store the SCs in the model $5
#echo T/Crev running ...
#cadre/tool_kit/crev -config $6 -model $5 -title $7 -call_tree $ctree_name
#echo End successfully !!!
# Thierry Le Gal         April 12, 1992
# go

# Shell script to run the trimming process between two levels of hierarchy in the call tree
/u/legal/twk/calltree/awk/start_level /u/legal/twk/calltree/ctree/appli_ctree proc_input 0 2
Appendix B

This appendix presents the programs used to create the ISCA and the data structure accessed by it. A description of the purpose of each function, the inputs and the outputs is given at the beginning of each program.

Files presented:

- makefile
- data_struct.h
- isca_struct.h
- main.c
- creation.c
- callbacks.c
- load_processes.c
# Thierry Le Gal May 14, 1992
# Makefile
# Makefile to produce the executable isca running the ISCA using the actual access to the database
# created by myself
# To create the executable : make
#

cflags= -I/usr/include/Xm
libs=-lXm -lX11

source=\
  /u/legal/ibm/isca/main.c\c
  /u/legal/ibm/isca/creation.c\c
  /u/legal/ibm/isca/callbacks.c\c
  /u/legal/ibm/isca/load_processes.c

objects=$(source: c= o)

all: isca

isca: $(objects)
    @echo "Creating executable .."
    @cc $(cflags) -o isca $(objects) $(libs)
    @echo "Type 'isca' to run"

.c.o:
    cc $(cflags) -c -g $*.c
#define IDENT_MAX 32
#define FILENAME 80

#define FTN 0 /* function kind */
#define IO 1 /* io library module kind */
#define LIB 2 /* regular library module kind */
#define VAR 3 /* global variable kind */

struct dbmddata_s {
    char fct_name[NUM];
    char type[NUM];
    int kind;
    char filename[NUM];
    int line_num;
    int param_count;
    int loglobalvar_count; /* Global variable accessed by the function */
    int call_fct_count;
    int call_lib_count;
    int call_io_count;
    int calledby_count;
    struct param_s *ptr_param;
    struct loglobalvar_s *ptr_loglobalvar;
    struct call_fct_s *ptr_call_fct;
    struct call_lib_s *ptr_call_lib;
    struct call_io_s *ptr_call_io;
    struct calledby_s *ptr_calledby;
    struct dbmddata_s *next;
};

struct param_s {
    char name[NUM];
    char type[NUM];
    struct param_s *next;
};

struct loglobalvar_s {
    char name[NUM];
    char type[NUM];
    struct loglobalvar_s *next;
};

struct call_fct_s {
    char name[NUM];
    struct call_fct_s *next;
};

struct call_lib_s {
    char name[NUM];
    struct call_lib_s *next;
};

struct call_io_s {
    char name[NUM];
    struct call_io_s *next;
};

struct calledby_s {
char name[FILENAME];
struct calledby_s *next;
};
/** Thierry Le Gal May 14, 1992 */
/* isca_struct.h */
/* Contains the global structures for the ISCA */
/* */

/* global structure for the module list widget */
typedef struct _Module_listStruct
{
    Widget list1;
} Module_listStruct;

/* global structure for the global variable list widget */
typedef struct _Global_var_listStruct
{
    Widget list1;
    Widget list2;
    Widget text1;
} Global_var_listStruct;

/* global structure for the module detail widget */
typedef struct _Module_detailStruct
{
    Widget list1;
    Widget list2;
    Widget list3;
    Widget list4;
    Widget text1;
    Widget text2;
    Widget text3;
} Module_detailStruct;

/* global structure for the source code widget */
typedef struct _Source_codeStruct
{
    Widget text1;
    Widget text2;
} Source_codeStruct;

/* structure containing all the structures for the ISCA */
typedef struct _Globals
{
    Module_detailStruct w3ids;
    Module_listStruct w1ids;
    Global_var_listStruct w2ids;
    Source_codeStruct w4ids;
} Globals;
/* Thierry Le Gal    May 14, 1992 */
/* main.c.c */
/* Main program for the ISCA */

/* Required MOTIF include files */
#include <Xm/Xm.h>
#include <X11/Shell.h>
#include <Xm/DialogS.h>

/* User include files */
#include "isca_struct.h"

/* Define section */
#define BX_APP_NAME "ISCA"
#define BX_APP_NAME1 "Module List"
#define BX_APP_NAME2 "Module Detail"
#define BX_APP_NAME3 "Source Code"
#define BX_APP_NAME4 "Global Variable List"
#define BX_APP_CLASS "BuilderProduct"

/* Shell callback procedure declarations */

/* Global widget variables */
Widget Shell000;
Widget Form;
Widget CreateForm();
Widget Shell001;
Widget Form1;
Widget CreateForm1();
Widget Shell002;
Widget Form2;
Widget CreateForm2();
Widget Shell003;
Widget Form3;
Widget CreateForm3();

/* Main program */
main(argc, argv)
int argc;
char **argv;
{
  Display *display;
  Arg args[256];
  int argcnt;
  XtAppContext context;
  Globals *globals;
  globals = (Globals *) XtAlloc(1, sizeof(Globals));

  XtToolkitInitialize();
  context = XtCreateApplicationContext();
  display = XtOpenDisplay(context, 0, BX_APP_NAME, BX_APP_CLASS,
  0, 0, &argc, argv);
  if(display == NULL)
    
    XtWarning("cannot open display");
    exit(1);
  }
  XmRegisterConverters();
  argcnt = 0;
  /* XtSetArg(args[argcnt], XmN, 22); argcnt++; */
XtSetArg(args[argcnt], XmNn, 84); argcnt++;
XtSetArg(args[argcnt], XmNn, -1000); argcnt++;
XtSetArg(args[argcnt], XmNn, -1000); argcnt++;
Shell000 = XtAppCreateShell( BX_APP_NAME1, BX_APP_CLASS,
applicationShellWidgetClass, display, args, argcnt);
argcnt = 0;
    /* XtSetArg(args[argcnt], XmNs, 20); argcnt++;
XtSetArg(args[argcnt], XmNs, 473); argcnt++;
XtSetArg(args[argcnt], XmNs, 1000); argcnt++;
XtSetArg(args[argcnt], XmNs, 1000); argcnt++;
Shell001 = XtAppCreateShell( BX_APP_NAME2, BX_APP_CLASS,
applicationShellWidgetClass, display, args, argcnt);
argcnt = 0;
    */
XtSetArg(args[argcnt], XmNs, 625); argcnt++;
XtSetArg(args[argcnt], XmNs, 81); argcnt++;
XtSetArg(args[argcnt], XmNs, 0); argcnt++;
XtSetArg(args[argcnt], XmNs, 0); argcnt++;
Shell002 = XtAppCreateShell( BX_APP_NAME3, BX_APP_CLASS,
applicationShellWidgetClass, display, args, argcnt);
argcnt = 0;
    */
XtSetArg(args[argcnt], XmNs, 267); argcnt++;
XtSetArg(args[argcnt], XmNs, 83); argcnt++;
XtSetArg(args[argcnt], XmNy, 0); argcnt++;
XtSetArg(args[argcnt], XmNy, 0); argcnt++;
Shell003 = XtAppCreateShell( BX_APP_NAME4, BX_APP_CLASS,
applicationShellWidgetClass, display, args, argcnt);
Form = CreateForm(Shell000, globals);
XtManageChild(Form);
Form1 = CreateForm1(Shell001, globals);
XtManageChild(Form1);
Form2 = CreateForm2(Shell002, globals);
XtManageChild(Form2);
Form3 = CreateForm3(Shell003, globals);
XtManageChild(Form3);
XtRealizeWidget(Shell000);
XtRealizeWidget(Shell001);
XtRealizeWidget(Shell002);
XtRealizeWidget(Shell003);

XtAppMainLoop(context);
/* Thierry Le Gal May 14, 1992 */
/* creation-c.c */
/* Creation file for the ISCA */

/* Required MOTIF include files */
#include <Xm/Xm.h>
#include <X11/Shell.h>
#include <Xm/ArrowB.h>
#include <Xm/ArrowBG.h>
#include <Xm/BulletinB.h>
#include <Xm/CascadeB.h>
#include <Xm/CascadeBG.h>
#include <Xm/Command.h>
#include <Xm/CutPaste.h>
#include <Xm/DialogS.h>
#include <Xm/DrawingA.h>
#include <Xm/DrawnB.h>
#include <Xm/FileSB.h>
#include <Xm/Form.h>
#include <Xm/Frame.h>
#include <Xm/Label.h>
#include <Xm/LabelG.h>
#include <Xm/List.h>
#include <Xm/MainW.h>
#include <Xm/MenuShell.h>
#include <Xm/MessageB.h>
#include <Xm/PaneW.h>
#include <Xm/PushB.h>
#include <Xm/PushBG.h>
#include <Xm/RowColumn.h>
#include <Xm/Scale.h>
#include <Xm/ScrollBar.h>
#include <Xm/ScrolledW.h>
#include <Xm/SelectoB.h>
#include <Xm/SeparatorG.h>
#include <Xm/Separator.h>
#include <Xm/Text.h>
#include <Xm/ToggleB.h>
#include <Xm/ToggleBG.h>

/* User supplied include files */
#include "isca_struct.h"

/* Some convenience routines */
#define IGNORE_COLOR
static unsigned long COLOR(w, name)
Widget w;
char *name;
{
XrmValue fromVal, toVal;
unsigned long *pixel;

fromVal.size = sizeof(char*);
fromVal.addr = name;

XtConvert(w, XmRString, &fromVal, XmRPixel, &toVal);
pixel = (unsigned long*)toVal.addr;
if( pixel == NULL )
{

}
fromVal.addr = XtDefaultBackground;
XtConvert(w, XmRString, &fromVal, XmRPixel, &toVal);
    pixel = (unsigned long*)toVal.addr;
}    
return("pixel");
}
#endif
#define IGNORE_FONT
static XmFontList
FONT_LIST(w, name)
Widget w;
char *name;
{
    XmValue fromVal, toVal;
    XmFontList *fontListPtr;
    
    fromVal.size = sizeof(char*);
    fromVal.addr = name;
    
    XtConvert(w, XmRString, &fromVal, XmRFontList, &toVal);
    fontListPtr = (XmFontList*)toVal.addr;
    if(fontListPtr == NULL)
    {
        fromVal.addr = "fixed";
        XtConvert(w, XmRString, &fromVal, XmRFontList, &toVal);
        fontListPtr = (XmFontList*)toVal.addr;
    }
    return(*fontListPtr);
}
#endif
#define IGNORE_MENU_POST
static void
MENU_POST(p, m, e)
Widget p;
Widget m;
XButtonEvent *e;
{
    Arg args[2];
    int argcnt;
    int button;
    
    argcnt = 0;
    XtSetArg(args[argcnt], XmNwhichButton, &button);
    argcnt++;
    XtGetValues(m, args, argcnt);
    if(e->button != button) return;
    XmMenuPosition(m, e);
    XtManageChild(m);
}
#endif
#define IGNORE_STRING_TABLE
#include <varargs.h>

static XmString*
STRING_TABLE(va_alist)
va_dcl
{
    va_list ap;
    int count;
    XmString *array;
    int i;
}
va_start(ap);
count = va_arg(ap, int);
array = (XmString*)XtMalloc((count + 1) * sizeof(XmString*));
for(i = 0; i < count; i++)
{
array[i] = XmStringCreateToR(va_arg(ap, char*),
    XmSTRING_DEFAULT_CHARSET);
}
array[count] = (XmString)0;
va_end(ap);
return(array);
}

static void
FREE_STRING_TABLE(table)
XmString *table;
{
int i;

for (i = 0; table[i]; i++)
{
    XmStringFree(table[i]);
}
    XtFree(table);
}
#endif

/* Callback procedure declarations */

/* Used for any widget */
void creation();
void find_list_item_callback();
void sort_list_callback();

/* Callbacks for the module list widget */
void w1_all_val_changed_callback();
void w1_priv_val_changed_callback();
void w1_lib_val_changed_callback();
void w1_io_val_changed_callback();
void w1_get_selected_list_item_callback();

/* Callbacks for the global variable list widget */
void w2_all_val_changed_callback();
void w2_loc_val_changed_callback();
void w2_get_selected_list_item_callback();

/* Callbacks for the source code widget */
void w4_load_callback();
void w4_exit_callback();
void w4_find_callback();

/*...........................................................................
 */
/* Module list widget */
/*...........................................................................
 */
Widget Createform(parent, globals)
Widget parent;
Globals *globals;
{
    Arg args[512];
    int argcnt;
    Widget retval;
XmString xmstr[32];
XmString *xstrTable[8];
Widget form;
Widget scrolledWindow;
Widget list1;
Widget frame2;
Widget pushButton;
Widget frame1;
Widget pushButton1;
Widget frame;
Widget radioBox;
Widget toggleButton;
Widget toggleButton1;
Widget toggleButton2;
Widget toggleButton3;
XmRegisterConverters();

argc = 0;
XtSetArg(args[argc], XmNx, 0); argc++;
XtSetArg(args[argc], XmNy, 0); argc++;
XtSetArg(args[argc], XmNwidth, 229); argc++;
XtSetArg(args[argc], XmNheight, 346); argc++;
form = XtCreateWidget("form",
xmFormWidgetClass,
parent,
args,
argc);

retval = form;

argc = 0;
XtSetArg(args[argc], XmNtopAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNbottomAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNleftAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNrightAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNtopPosition, (int)2); argc++;
XtSetArg(args[argc], XmNbottomPosition, (int)9); argc++;
XtSetArg(args[argc], XmNleftPosition, (int)5); argc++;
XtSetArg(args[argc], XmNrightPosition, (int)95); argc++;
XtSetArg(args[argc], XmNx, 11); argc++;
XtSetArg(args[argc], XmNy, 17); argc++;
XtSetArg(args[argc], XmNwidth, 207); argc++;
XtSetArg(args[argc], XmNheight, 25); argc++;
frame = XtCreateWidget("frame",
xmFrameWidgetClass,
form,
args,
argc);

XtManageChild(frame);

argc = 0;
XtSetArg(args[argc], XmNtopAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNbottomAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNleftAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNrightAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNtopPosition, (int)92); argc++;
XtSetArg(args[argc], XmNbottomPosition, (int)98); argc++;
XtSetArg(args[argc], XmNleftPosition, (int)73); argc++;
XtSetArg(args[argc], XmNrightPosition, (int)98); argc++;
XtSetArg(args[argc], XmNx, 160); argc++;
XtSetArg(args[argc], XmNy, 301); argc++;
XtSetArg(args[argcnt], XmNwidth, 58); argcnt++;
XtSetArg(args[argcnt], XmNheight, 25); argcnt++;
frame1 = XtCreateWidget("frame1",
xmFrameWidgetClass,
form,
args,
argcnt);

XtManageChild(frame1);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)92); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)98); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)2); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)27); argcnt++;
XtSetArg(args[argcnt], XmNx, 11); argcnt++;
XtSetArg(args[argcnt], XmNy, 301); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 58); argcnt++;
XtSetArg(args[argcnt], XmNheight, 25); argcnt++;
frame2 = XtCreateWidget("frame2",
xmFrameWidgetClass,
form,
args,
argcnt);

XtManageChild(frame2);

argcnt = 0;
XtSetArg(args[argcnt], XmNscrollingPolicy, XmAUTOMATIC); argcnt++;
XtSetArg(args[argcnt], XmNscrollBarDisplayPolicy, XmSTATIC); argcnt++;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)11); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)90); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)5); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)95); argcnt++;
XtSetArg(args[argcnt], XmNx, 11); argcnt++;
XtSetArg(args[argcnt], XmNy, 52); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 207); argcnt++;
XtSetArg(args[argcnt], XmNheight, 242); argcnt++;
scrolledWindow = XtCreateWidget("scrolledWindow",
xmScrolledWindowWidgetClass,
form,
args,
argcnt);

XtManageChild(scrolledWindow);

argcnt = 0;
" XSetArg(args[argcnt], XmNitems, (xmstrTable[0]=STRING_TABLE(25, "item1",
"item2",
"3",
"4",
"5",
"6",
"7",
"8",

132
xmPushButtonWidgetClass,
frame1,
args,
argcnt);
XmStringFree( xmstr[0] );

XtManageChild(pushButton1);

argcnt = 0;
XtSetArg(args[argcnt], XmNradioBehavior, True); argcnt++;
XtSetArg(args[argcnt], XmNradioAlwaysOn, True); argcnt++;
XtSetArg(args[argcnt], XmNpacking, XmPACK_COLUMN); argcnt++;
XtSetArg(args[argcnt], XmNorientation, XmHORIZONTAL); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 203); argcnt++;
XtSetArg(args[argcnt], XmNheight, 21); argcnt++;
radioBox = XtCreateWidget("radioBox",
xmRowColumnWidgetClass,
frame,
args,
argcnt);

XtManageChild(radioBox);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR("All",
XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNx, 3); argcnt++;
XtSetArg(args[argcnt], XmNy, 3); argcnt++;
toggleButton = XtCreateWidget("ToggleButton",
xmToggleButtonWidgetClass,
radioBox,
args,
argcnt);
XmStringFree( xmstr[0] );
XtAddCallback(toggleButton, XmNvalueChangedCallback, w1_all_val_changed_callback, (caddr_t)globals);

XtManageChild(toggleButton);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Priv",
XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNx, 51); argcnt++;
XtSetArg(args[argcnt], XmNy, 3); argcnt++;
toggleButton1 = XtCreateWidget("ToggleButton1",
xmToggleButtonWidgetClass,
radioBox,
args,
argcnt);
XmStringFree( xmstr[0] );
XtAddCallback(toggleButton1, XmNvalueChangedCallback, w1_priv_val_changed_callback, (caddr_t)globals);

XtManageChild(toggleButton1);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Lib",
XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNx, 99); argcnt++;
XtSetArg(argv[argcnt], XmNy, 3); argcnt++;
toggleButton2 = XtCreateWidget("toggleButton2",
XmToggleButtonWidgetClass,
radioBox,
argv,
argcnt);
XmStringFree( xstr[0] );
 XtAddCallback(toggleButton2, XmNvalueChangedCallback, w1_lib_val_changed_callback, (caddr_t)globals);
 XtManageChild(toggleButton2);

argcnt = 0;
XtSetArg(argv[argcnt], XmNlabelString, (xstr[0]=XmStringCreateLtoR("I/O",
XmSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg(argv[argcnt], XmNx, 147); argcnt++;
XtSetArg(argv[argcnt], XmNy, 3); argcnt++;
toggleButton3 = XtCreateWidget("toggleButton3",
XmToggleButtonWidgetClass,
radioBox,
argv,
argcnt);
XmStringFree( xstr[0] );
 XtAddCallback(toggleButton3, XmNvalueChangedCallback, w1_io_val_changed_callback, (caddr_t)globals);
 XtManageChild(toggleButton3);

return( retval );
}

/*............................................................................
" Module detail widget

*/
/*............................................................................*/

Widget Createform1(parent, globals)
    Widget parent;
    Globals *globals;
{
    Arg argv[512];
    int argcnt;
    Widget retval;
    XmString xstr[32];
    XmString *xstrTable[8];
    Widget form1;
    Widget scrolledWindow4;
    Widget list2;
    Widget frame15;
    Widget label;
    Widget frame14;
    Widget label1;
    Widget scrolledWindow3;
    Widget list3;
    Widget frame13;
    Widget pushButton2;
    Widget frame12;
    Widget pushButton3;
    Widget frame11;
    Widget pushButton4;
    Widget scrolledWindow2;
    Widget list4;
    Widget frame10;
    Widget label2;
    Widget scrolledWindow1;
    Widget list5;
Widget frame9;
Widget label3;
Widget frame8;
Widget text;
Widget frame7;
Widget text1;
Widget frame6;
Widget text2;
Widget frame5;
Widget label4;
Widget frame4;
Widget label5;
Widget frame3;
Widget label6;
XmRegisterConverters();

argc = 0;
XtSetArg(args[argc], XmNx, 0); argc++;
XtSetArg(args[argc], XmNy, 0); argc++;
XtSetArg(args[argc], XmNwidth, 589); argc++;
XtSetArg(args[argc], XmNheight, 526); argc++;
form1 = XtCreateWidget("form1",
xmFormWidgetClass,
parent,
args,
argc);

retval = form1;

argc = 0;
XtSetArg(args[argc], XmNtopAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNbottomAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNleftAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNrightAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNtopPosition, (int)25); argc++;
XtSetArg(args[argc], XmNbottomPosition, (int)30); argc++;
XtSetArg(args[argc], XmNleftPosition, (int)3); argc++;
XtSetArg(args[argc], XmNrightPosition, (int)13); argc++;
XtSetArg(args[argc], XmNx, 18); argc++;
XtSetArg(args[argc], XmNy, 132); argc++;
XtSetArg(args[argc], XmNwidth, 59); argc++;
XtSetArg(args[argc], XmNheight, 28); argc++;
frame3 = XtCreateWidget("frame3",
xmFrameWidgetClass,
form1,
args,
argc);

XtManageChild(frame3);

argc = 0;
XtSetArg(args[argc], XmNtopAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNbottomAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNleftAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNrightAttachment, XmATTACH_POSITION); argc++;
XtSetArg(args[argc], XmNtopPosition, (int)16); argc++;
XtSetArg(args[argc], XmNbottomPosition, (int)21); argc++;
XtSetArg(args[argc], XmNleftPosition, (int)3); argc++;
XtSetArg(args[argc], XmNrightPosition, (int)13); argc++;
XtSetArg(args[argc], XmNx, 18); argc++;
XtSetArg(args[argc], XmNy, 84); argc++;
XtSetArg(args[argc], XmNwidth, 59); argc++;
XtSetArg(args[argc], XmNheight, 28); argc++;

136
frame4 = XtCreateWidget("frame4",
xmFrameWidgetClass,
form1,
args,
argcnt);
XtManageChild(frame4);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)8); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)13); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)3); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)13); argcnt++;
XtSetArg(args[argcnt], XmNX, 18); argcnt++;
XtSetArg(args[argcnt], XmNY, 42); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 59); argcnt++;
XtSetArg(args[argcnt], XmNheight, 26); argcnt++;
frame5 = XtCreateWidget("frame5",
xmFrameWidgetClass,
form1,
args,
argcnt);
XtManageChild(frame5);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)25); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)30); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)15); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNX, 88); argcnt++;
XtSetArg(args[argcnt], XmNY, 132); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 195); argcnt++;
XtSetArg(args[argcnt], XmNheight, 26); argcnt++;
frame6 = XtCreateWidget("frame6",
xmFrameWidgetClass,
form1,
args,
argcnt);
XtManageChild(frame6);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)16); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)21); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)15); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNX, 88); argcnt++;
XtSetArg(args[argcnt], XmNY, 84); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 195); argcnt++;
XtSetArg(args[argcnt], XmNheight, 26); argcnt++;
frame7 = XtCreateWidget("frame7",
xmFrameWidgetClass,
form1,
args,
argcnt);

XtManageChild(frame7);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)8); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)13); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)15); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmN, 88); argcnt++;
XtSetArg(args[argcnt], XmNy, 42); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 195); argcnt++;
XtSetArg(args[argcnt], XmNheight, 26); argcnt++;
frame8 = XtCreateWidget("frame8",
xmFrameWidgetClass,
form1,
args,
argcnt);

XtManageChild(frame8);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)37); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)42); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)3); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNx, 18); argcnt++;
XtSetArg(args[argcnt], XmNy, 195); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 268); argcnt++;
XtSetArg(args[argcnt], XmNheight, 26); argcnt++;
frame9 = XtCreateWidget("frame9",
xmFrameWidgetClass,
form1,
args,
argcnt);

XtManageChild(frame9);

argcnt = 0;
XtSetArg(args[argcnt], XmNscrollingPolicy, XmAUTOMATIC); argcnt++;
XtSetArg(args[argcnt], XmNscrollBarDisplayPolicy, XmDYNAMIC); argcnt++;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)43); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)65); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)3); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNx, 19); argcnt++;
XtSetArg(args[argcnt], XmNy, 226); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 265); argcnt++;
XtSetArg(args[argcnt], XmNheight, 116); argcnt++;
scrolledWindow1 = XtCreateWidget("scrolledWindow1",
xmScrolledWindowWidgetClass,
form1,
args,
argcnt);

XtManageChild(scrolledWindow1);

argcnt = 0;
XtSetArg(args[argcnt], XmNTopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)67); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)72); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)3); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNX, 18); argcnt++;
XtSetArg(args[argcnt], XmNY, 352); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 265); argcnt++;
XtSetArg(args[argcnt], XmNheight, 27); argcnt++;
frame10 = XtCreateWidget("frame10",
xmFrameWidgetClass,
form1,
args,
argcnt);

XtManageChild(frame10);

argcnt = 0;
XtSetArg(args[argcnt], XmNscrollingPolicy, XmAUTOMATIC); argcnt++;
XtSetArg(args[argcnt], XmNscrollBarDisplayPolicy, XmDYNAMIC); argcnt++;
XtSetArg(args[argcnt], XmNTopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)73); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)97); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)3); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNX, 18); argcnt++;
XtSetArg(args[argcnt], XmNY, 384); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 265); argcnt++;
XtSetArg(args[argcnt], XmNheight, 126); argcnt++;
scrolledWindow2 = XtCreateWidget("scrolledWindow2",
xmScrolledWindowWidgetClass,
form1,
args,
argcnt);

XtManageChild(scrolledWindow2);

argcnt = 0;
XtSetArg(args[argcnt], XmNTopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)43); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)52); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)67); argcnt++;

139
XtSetArg(args[argcnt], XmNx, 306); argcnt++;
XtSetArg(args[argcnt], XmNy, 225); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 88); argcnt++;
XtSetArg(args[argcnt], XmNheight, 26); argcnt++;
frame11 = XtCreateWidget("frame11",
xmFrameWidgetClass,
form1,
args,
argcnt);

XtManageChild(frame11);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)93); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)98); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)82); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)97); argcnt++;
XtSetArg(args[argcnt], XmNx, 483); argcnt++;
XtSetArg(args[argcnt], XmNy, 489); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 88); argcnt++;
XtSetArg(args[argcnt], XmNheight, 26); argcnt++;
frame12 = XtCreateWidget("frame12",
xmFrameWidgetClass,
form1,
args,
argcnt);

XtManageChild(frame12);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)93); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)98); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)52); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)67); argcnt++;
XtSetArg(args[argcnt], XmNx, 306); argcnt++;
XtSetArg(args[argcnt], XmNy, 488); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 89); argcnt++;
XtSetArg(args[argcnt], XmNheight, 25); argcnt++;
frame13 = XtCreateWidget("frame13",
xmFrameWidgetClass,
form1,
args,
argcnt);

XtManageChild(frame13);

argcnt = 0;
XtSetArg(args[argcnt], XmNscrollingPolicy, XmAUTOMATIC); argcnt++;
XtSetArg(args[argcnt], XmNvisualPolicy, XmCONSTANT); argcnt++;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)58); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)91); argcnt++;
XtSetArg(args[argc], XmNlightPosition, (int)52); argcnt++;
XtSetArg(args[argc], XmNrightPosition, (int)97); argcnt++;
XtSetArg(args[argc], XmNtopOffset, (int)0); argcnt++;
XtSetArg(args[argc], XmNx, 306); argcnt++;
XtSetArg(args[argc], XmNy, 305); argcnt++;
XtSetArg(args[argc], XmNwidth, 265); argcnt++;
XtSetArg(args[argc], XmNheight, 174); argcnt++;
scrolledWindow3 = XtCreateWidget("scrolledWindow3",
XmScrolledWindowWidgetClass,
form1,
args,
argcnt);
XtManageChild(scrolledWindow3);
argcnt = 0;
XtSetArg(args[argc], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNtopPosition, (int)52); argcnt++;
XtSetArg(args[argc], XmNbottomPosition, (int)57); argcnt++;
XtSetArg(args[argc], XmNleftPosition, (int)52); argcnt++;
XtSetArg(args[argc], XmNrightPosition, (int)87); argcnt++;
XtSetArg(args[argc], XmNx, 306); argcnt++;
XtSetArg(args[argc], XmNy, 274); argcnt++;
XtSetArg(args[argc], XmNwidth, 265); argcnt++;
XtSetArg(args[argc], XmNheight, 26); argcnt++;
frame14 = XtCreateWidget("frame14",
XmFrameWidgetClass,
form1,
args,
argcnt);
XtManageChild(frame14);
argcnt = 0;
XtSetArg(args[argc], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNtopPosition, (int)2); argcnt++;
XtSetArg(args[argc], XmNbottomPosition, (int)7); argcnt++;
XtSetArg(args[argc], XmNleftPosition, (int)52); argcnt++;
XtSetArg(args[argc], XmNrightPosition, (int)7); argcnt++;
XtSetArg(args[argc], XmNx, 306); argcnt++;
XtSetArg(args[argc], XmNy, 11); argcnt++;
XtSetArg(args[argc], XmNwidth, 265); argcnt++;
XtSetArg(args[argc], XmNheight, 26); argcnt++;
frame15 = XtCreateWidget("frame15",
XmFrameWidgetClass,
form1,
args,
argcnt);
XtManageChild(frame15);
argcnt = 0;
XtSetArg(args[argc], XmNmappedWhen Managed, True); argcnt++;
XtSetArg(args[argc], XmNscrollingPolicy, XmAUTOMATIC); argcnt++;
XtSetArg(args[argc], XmNvisualPolicy, XmCONSTANT); argcnt++;
XtSetArg(args[argc], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argc], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;

141
XtSetArg(args[argc], XmNleftAttachment, XmATTACH_POSITION); argc++;  
XtSetArg(args[argc], XmNrightAttachment, XmATTACH_POSITION); argc++;  
XtSetArg(args[argc], XmNtopPosition, (int)8); argc++;  
XtSetArg(args[argc], XmNbottomPosition, (int)42); argc++;  
XtSetArg(args[argc], XmNleftPosition, (int)52); argc++;  
XtSetArg(args[argc], XmNrightPosition, (int)87); argc++;  
XtSetArg(args[argc], XmNX, 306); argc++;  
XtSetArg(args[argc], XmNY, 42); argc++;  
XtSetArg(args[argc], XmNwidth, 265); argc++;  
XtSetArg(args[argc], XmNheight, 179); argc++;  
scrolledWindow4 = XtCreateWidget("scrolledWindow4", 
xmScrolledWindowWidgetClass, 
form1,  
args,  
argc);  
XtManageChild(scrolledWindow4);  
/* XtSetArg(args[argc], XmNitems, (xstrTable[0]=STRING_TABLE(3, "jhfghdjhgfdfhgdjghjdjdg", 
"hfsghkh",  
"lkjlkjdsl")); argc++;  
XtSetArg(args[argc], XmNitemCount, (int)3); argc++; */  
XtSetArg(args[argc], XmNitemCount, (int)0); argc++;  
XtSetArg(args[argc], XmNvisibleItemCount, (int)100); argc++;  
XtSetArg(args[argc], XmNselectionPolicy, XmSINGLE_SELECT); argc++;  
XtSetArg(args[argc], XmNlistSizePolicy, XmCONSTANT); argc++;  
XtSetArg(args[argc], XmNscrollBarDisplayPolicy, XmSTATIC); argc++;  
XtSetArg(args[argc], XmNX, 0); argc++;  
XtSetArg(args[argc], XmNY, 0); argc++;  
XtSetArg(args[argc], XmNwidth, 500); argc++;  
list2 = XtCreateWidget("list2",  
xmListWidgetClass,  
scrolledWindow4,  
args,  
argc);  
FREE_STRING_TABLE(xstrTable[0]);  
creation(list2,(caddr_t)&(globals->w3ids.list3),(caddr_t) 0);  
XtManageChild(list2);  
argc = 0;  
XtSetArg(args[argc], XmNalignment, XmALIGNMENT_BEGINNING); argc++;  
XtSetArg(args[argc], XmNlabelString, (xstr[0]=xStringCreateLoR("Calls:"),  
XmSTRING_DEFAULT_CHARSET))); argc++;  
XtSetArg(args[argc], XmNnrecomputeSize, True); argc++;  
XtSetArg(args[argc], XmNX, 2); argc++;  
XtSetArg(args[argc], XmNY, 2); argc++;  
label = XtCreateWidget("label",  
xmLabelWidgetClass,  
frame15,  
args,  
argc);  
XmStringFree( xstr[0] );  
XtManageChild(label);  
argc = 0;  
XtSetArg(args[argc], XmNalignment, XmALIGNMENT_BEGINNING); argc++;  
XtSetArg(args[argc], XmNlabelString, (xstr[0]=xStringCreateLoR("Called by:"),  
XmSTRING_DEFAULT_CHARSET))); argc++;
XtSetArg(args[argcnt], XmNrecomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNxs, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
lbl1 = XtCreateWidget("label1", x11LabelWidgetClass,
frame14,
args,
argcnt);
XmStringFree(xmstr[0]);
XtManageChild(lbl1);

argcnt = 0;
XtSetArg(args[argcnt], XmNsensitive, True); argcnt++;
XtSetArg(args[argcnt], XmNmappedWhenManaged, True); argcnt++;
/* XtSetArg(args[argcnt], XmNitems, (xmstrTable[0].STRING_TABLE(7, "iuf",
"1kjd1jkd",
"jnsdkjksd",
"kjkj",
"hikdhdj",
"kkhjksd",
"hjk1kj3k44k55k66k67k68k69k6a")); argcnt++;
XtSetArg(args[argcnt], XmNitemCount, (int)6); argcnt++;
XtSetArg(args[argcnt], XmNitemCount, (int)0); argcnt++;
XtSetArg(args[argcnt], XmNvisibleItemCount, (int)100); argcnt++;
XtSetArg(args[argcnt], XmNselectionPolicy, XmSINGLE_SELECT); argcnt++;
XtSetArg(args[argcnt], XmNcallbackPolicy, XmCONSTANT); argcnt++;
XtSetArg(args[argcnt], XmNscrollBarDisplayPolicy, XmSTATIC); argcnt++;
XtSetArg(args[argcnt], XmNautomaticSelection, False); argcnt++;
XtSetArg(args[argcnt], XmNx, 0); argcnt++;
XtSetArg(args[argcnt], XmNy, 0); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 500); argcnt++;
list3 = XtCreateWidget("list3", x11ListWidgetClass,
scrolledWindow3,
args,
argcnt);
FREE_STRING_TABLE(xmstrTable[0]);
creation(list3,(caddr_t)&(globals->w3ids.list4),(caddr_t)0);
XtManageChild(list3);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Find",
XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
pushButton2 = XtCreateWidget("pushButton2", x11PushButtonWidgetClass,
frame13,
args,
argcnt);
XmStringFree(xmstr[0]);

/* XtAddCallback(pushButton2, XmNactivateCallback, find_bottom_list_callback, (caddr_t)0);*/
XtManageChild(pushButton2);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Exit",
XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
pushButton3 = XtCreateWidget("pushButton3",
xmPushButtonWidgetClass,
frame12,
args,
argcnt);
XmStringFree(xmstr[0]);

/* XtAddCallback(pushButton3, XmNactivateCallback, exit_callback, (caddr_t)0); */
XtManageChild(pushButton3);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLoR(
"Find",
XmSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
pushButton4 = XtCreateWidget("pushButton4",
xmPushButtonWidgetClass,
frame11,
args,
argcnt);
XmStringFree(xmstr[0]);

/* XtAddCallback(pushButton4, XmNactivateCallback, find_top_list_callback, (caddr_t)0); */
XtManageChild(pushButton4);

argcnt = 0;
/* XtSetArg(args[argcnt], XmNitems, (xmstrTable[0]=STRING_TABLE(6, "ijklhijds",
"gldcba", "jdf",
"hghsdagfsda",
"lknd", )); argcnt++;
XtSetArg(args[argcnt], XmNitemCount, (int)6); argcnt++; */
XtSetArg(args[argcnt], XmNitemCount, (int)0); argcnt++;
XtSetArg(args[argcnt], XmNvisibleItemCount, (int)0); argcnt++;
XtSetArg(args[argcnt], XmNselectionPolicy, XmSINGLE_SELECT); argcnt++;
XtSetArg(args[argcnt], XmNlistSizePolicy, XmCONSTANT); argcnt++;
XtSetArg(args[argcnt], XmNscrollBarDisplayPolicy, XmSTATIC); argcnt++;
XtSetArg(args[argcnt], XmNx, 0); argcnt++;
XtSetArg(args[argcnt], XmNy, 0); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 500); argcnt++;
list4 = XtCreateWidget("list4",
xmListWidgetClass,
scrolledWindow2,
args,
argcnt);
FREE_STRING_TABLE(xmstrTable[0]);

creation(list4,(caddr_t)&(globals->w3ids.list2),(caddr_t) 0);
XtManageChild(list4);

argcnt = 0;
XtSetArg(args[argcnt], XmNalignment, XmALIGNMENT_BEGINNING); argcnt++;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLoR(
"Global variables:",
XmSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
label2 = XmCreateWidget("label2",
XmLabelWidgetClass,
frame10,
args,
argcnt);
XmStringFree( xmstr[0] );

XtManageChild(label2);

argcnt = 0;
/* XtSetArg(args[argcnt], XmNitems, (xmstrTable[0]=STRING_TABLE(5, "jbdhhjiijj",
"khlijhjkhd", "khjgdsnkjhd", "ks7316679083ijkh",
"lkstd")); argcnt++; */
XtSetArg(args[argcnt], XmNitemCount, (int)5); argcnt++;
/* XtSetArg(args[argcnt], XmNitemCount, (int)0); argcnt++; */
XtSetArg(args[argcnt], XmNvisibleItemCount, (int)100); argcnt++;
XtSetArg(args[argcnt], XmNselectionPolicy, XmSINGLE_SELECT); argcnt++;
XtSetArg(args[argcnt], XmNlistSizePolicy, XmCONSTANT); argcnt++;
XtSetArg(args[argcnt], XmNscrollBarDisplayPolicy, XmSTATIC); argcnt++;
XtSetArg(args[argcnt], XmNnx, 0); argcnt++;
XtSetArg(args[argcnt], XmNny, 0); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 654); argcnt++;
list5 = XtCreateWidget("list5",
XmListBoxWidgetClass,
scrolledWindow1,
args,
argcnt);
FREE_STRING_TABLE( xmstrTable[0] );

creation((list5,(caddr_t)&globals->w3ids.list1),(caddr_t) 0);
XtManageChild(list5);

argcnt = 0;
XtSetArg(args[argcnt], XmNalignment, XmALIGNMENT_BEGINNING); argcnt++;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR( "Parameters:",
XmSTRING_DEFAULT_CHARSET])); argcnt++;
XtSetArg(args[argcnt], XmNcomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
label3 = XtCreateWidget("label3",
XmLabelWidgetClass,
frame9,
args,
argcnt);
XmStringFree( xmstr[0] );

XtManageChild(label3);

argcnt = 0;
/* XtSetArg(args[argcnt], XmNvalue, "initial_name"); argcnt++; */
XtSetArg(args[argcnt], XmNmarginHeight, (short)1); argcnt++;
XtSetArg(args[argcnt], XmNfontList, FONT_LIST(parent, 
"-misc-fixed-medium-r-normal-10-100-75-75-+60-iso8859-1")); argcnt++;
XtSetArg(args[argcnt], XmNeditMode, XmSINGLE_LINE_EDIT); argcnt++;
XtSetArg(args[argcnt], XmNeditable, False); argcnt++;
XtSetArg(args[argcnt], XmNcursorPositionVisible, False); argcnt++;
XtSetArg(args[argcnt], XmNcolumns, (short)37); argcnt++;
XtSetArg(args[argcnt], XmNrows, (short)1); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;

145
XtSetArg[&sqlits[sqlit], XmNy, 2]; argcnt++,
text = XtCreateWidget("text",
xmTextWidgetClass,
frame8,
args,
argcnt);

creation(text,(caddr_t&(globals->w3ids.text1),(caddr_t)0);
XtManageChild(text);

argcnt = 0;

/* XtSetArg[&sqlits[sqlit], XmValue, "initial_type"); argcnt++;;/*
XtSetArg[&sqlits[sqlit], XmMarginHeight, (short)1]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNonList, FONT_LIST(parent, \n"*misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1")]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNeditable, False]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNcursorPositionVisible, False]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNcolumns, (short)37]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNrows, (short)1]; argcnt++;
XtSetArg[&sqlits[sqlit], XmN, 2]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNy, 2]; argcnt++;
text1 = XtCreateWidget("text1",
xmTextWidgetClass,
frame7,
args,
argcnt);

creation(text1,(caddr_t&(globals->w3ids.text2),(caddr_t)0);
XtManageChild(text1);

argcnt = 0;

/* XtSetArg[&sqlits[sqlit], XmValue, "initial_path"); argcnt++;;/*
XtSetArg[&sqlits[sqlit], XmMarginHeight, (short)1]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNonList, FONT_LIST(parent, \n"*misc-fixed-medium-r-normal--10-100-75-75-c-60-iso8859-1")]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNeditable, False]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNcursorPositionVisible, False]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNcolumns, (short)37]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNrows, (short)1]; argcnt++;
XtSetArg[&sqlits[sqlit], XmN, 2]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNy, 2]; argcnt++;
text2 = XtCreateWidget("text2",
xmTextWidgetClass,
frame6,
args,
argcnt);

creation(text2,(caddr_t&(globals->w3ids.text3),(caddr_t)0);
XtManageChild(text2);

argcnt = 0;
XtSetArg[&sqlits[sqlit], XmNalignment, XmALIGNMENT_CENTER]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNlabelString, (xstr[0]=XmStringCreateLocal("Name: ",
XMSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg[&sqlits[sqlit], XmNrecomputeSize, True]; argcnt++;
XtSetArg[&sqlits[sqlit], XmN, 2]; argcnt++;
XtSetArg[&sqlits[sqlit], XmNy, 2]; argcnt++;
label4 = XtCreateWidget("label4",
xmLabelWidgetClass,
frame5,
args,
argcnt);
XmStringFree(xmstr[0]);

 XtManageChild(label4);

 argcnt = 0;
 XtSetArg(argcnt, XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Type:",
 XmSTRING_DEFAULT_CHARSET())); argcnt++;
 XtSetArg(argcnt, XmNrecomputeSize, True); argcnt++;
 XtSetArg(argcnt, XmNx, 2); argcnt++;
 XtSetArg(argcnt, XmNy, 2); argcnt++;
 label5 = XmCreateWidget("label5",
xmLabelWidgetClass,
frame4,
args,
argcnt);
XmStringFree(xmstr[0]);

 XtManageChild(label5);

 argcnt = 0;
 XtSetArg(argcnt, XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Locat:",
 XmSTRING_DEFAULT_CHARSET())); argcnt++;
 XtSetArg(argcnt, XmNrecomputeSize, True); argcnt++;
 XtSetArg(argcnt, XmNx, 2); argcnt++;
 XtSetArg(argcnt, XmNy, 2); argcnt++;
 label6 = XmCreateWidget("label6",
xmLabelWidgetClass,
frame3,
args,
argcnt);
XmStringFree(xmstr[0]);

 XtManageChild(label6);

 return(retval);
}

/*.............................................................................*/
/* Global variables list widget */
/*.............................................................................*/
Widget Createform3(parent, globals)
Widget parent;
Globals *globals;
{
 Arg argcnt[512];
 int argcnt;
 Widget retalv;
 XmString xmstr[32];
 XmString *xmstrTable[8];
 Widget form3;
 Widget scrolledWindow7;
 Widget ist6;
 Widget scrolledWindow6;
 Widget list7;
 Widget frame26;
 Widget radioBox1;
 Widget toggleButton4;
 Widget toggleButton5;
 Widget frame25;
 Widget label8;
Widget frame24;
Widget text5;
Widget frame23;
Widget label9;
Widget frame22;
Widget pushButton8;
Widget frame21;
Widget pushButton9;
XmRegisterConverters();

argcnt = 0;
XtSetArg(argv[argcnt], XmNnx, 0); argcnt++;
XtSetArg(argv[argcnt], XmNny, 0); argcnt++;
XtSetArg(argv[argcnt], XmNwidth, 341); argcnt++;
XtSetArg(argv[argcnt], XmNheight, 347); argcnt++;
form3 = XtCreateWidget("form3",
parent,
args,
argcnt);

retval = form3;

argcnt = 0;
XtSetArg(argv[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNtopPosition, (int)92); argcnt++;
XtSetArg(argv[argcnt], XmNbottomPosition, (int)98); argcnt++;
XtSetArg(argv[argcnt], XmNleftPosition, (int)48); argcnt++;
XtSetArg(argv[argcnt], XmNrightPosition, (int)68); argcnt++;
XtSetArg(argv[argcnt], XmNnx, 164); argcnt++;
XtSetArg(argv[argcnt], XmNny, 319); argcnt++;
XtSetArg(argv[argcnt], XmNwidth, 68); argcnt++;
XtSetArg(argv[argcnt], XmNheight, 21); argcnt++;
frame21 = XtCreateWidget("frame21",
xmFrameWidgetClass,
form3,
args,
argcnt);

XtManageChild(frame21);

argcnt = 0;
XtSetArg(argv[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argv[argcnt], XmNtopPosition, (int)92); argcnt++;
XtSetArg(argv[argcnt], XmNbottomPosition, (int)98); argcnt++;
XtSetArg(argv[argcnt], XmNleftPosition, (int)22); argcnt++;
XtSetArg(argv[argcnt], XmNrightPosition, (int)22); argcnt++;
XtSetArg(argv[argcnt], XmNnx, 7); argcnt++;
XtSetArg(argv[argcnt], XmNny, 319); argcnt++;
XtSetArg(argv[argcnt], XmNwidth, 68); argcnt++;
XtSetArg(argv[argcnt], XmNheight, 21); argcnt++;
frame22 = XtCreateWidget("frame22",
xmFrameWidgetClass,
form3,
args,
argcnt);
XtManageChild(frame22);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)11); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)17); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)72); argcnt++;
XtSetArg(args[argcnt], XmNx, 164); argcnt++;
XtSetArg(args[argcnt], XmNy, 38); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 92); argcnt++;
XtSetArg(args[argcnt], XmNheight, 21); argcnt++;
frame23 = XtCreateWidget("frame23",
form3,
args,
argcnt);

XtManageChild(frame23);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)2); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)10); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)65); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)98); argcnt++;
XtSetArg(args[argcnt], XmNx, 222); argcnt++;
XtSetArg(args[argcnt], XmNy, 7); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 112); argcnt++;
XtSetArg(args[argcnt], XmNheight, 28); argcnt++;
frame24 = XtCreateWidget("frame24",
form3,
args,
argcnt);

XtManageChild(frame24);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)4); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)10); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)48); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)63); argcnt++;
XtSetArg(args[argcnt], XmNx, 164); argcnt++;
XtSetArg(args[argcnt], XmNy, 14); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 51); argcnt++;
XtSetArg(args[argcnt], XmNheight, 21); argcnt++;
frame25 = XtCreateWidget("frame25",
form3,
args,
argcnt);
XtManageChild(frame25);

argcnt = 0;
XtSetArg(argcnt, XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNtopPosition, (int)2); argcnt++;
XtSetArg(argcnt, XmNbottomPosition, (int)9); argcnt++;
XtSetArg(argcnt, XmNleftPosition, (int)2); argcnt++;
XtSetArg(argcnt, XmNrightPosition, (int)35); argcnt++;
XtSetArg(argcnt, XmNx, 7); argcnt++;
XtSetArg(argcnt, XmNy, 7); argcnt++;
XtSetArg(argcnt, XmNwidth, 95); argcnt++;
XtSetArg(argcnt, XmNheight, 24); argcnt++;
frame26 = XtCreateWidget("frame26",
xmFrameWidgetClass,
form3,
args,
argcnt);

XtManageChild(frame26);

argcnt = 0;
XtSetArg(argcnt, XmNsrollingPolicy, XmAUTOMATIC); argcnt++;
XtSetArg(argcnt, XmNsrollingBarDisplayPolicy, XmSTATIC); argcnt++;
XtSetArg(argcnt, XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNtopPosition, (int)48); argcnt++;
XtSetArg(argcnt, XmNbottomPosition, (int)88); argcnt++;
XtSetArg(argcnt, XmNx, 164); argcnt++;
XtSetArg(argcnt, XmNy, 62); argcnt++;
XtSetArg(argcnt, XmNwidth, 170); argcnt++;
XtSetArg(argcnt, XmNheight, 250); argcnt++;
scrolledWindow6 = XtCreateWidget("scrolledWindow6",
xmScrolledWindowWidgetClass,
form3,
args,
argcnt);

XtManageChild(scrolledWindow6);

argcnt = 0;
XtSetArg(argcnt, XmNsrollingPolicy, XmAUTOMATIC); argcnt++;
XtSetArg(argcnt, XmNsrollingBarDisplayPolicy, XmSTATIC); argcnt++;
XtSetArg(argcnt, XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(argcnt, XmNtopPosition, (int)11); argcnt++;
XtSetArg(argcnt, XmNbottomPosition, (int)89); argcnt++;
XtSetArg(argcnt, XmNleftPosition, (int)2); argcnt++;
XtSetArg(argcnt, XmNrightPosition, (int)45); argcnt++;
XtSetArg(argcnt, XmNx, 7); argcnt++;
XtSetArg(argcnt, XmNy, 38); argcnt++;
XtSetArg(argcnt, XmNwidth, 146); argcnt++;
XtSetArg(argcnt, XmNheight, 271); argcnt++;
scrolledWindow7 = XtCreateWidget("scrolledWindow7",
xmScrolledWindowWidgetClass,
xmRowColumnWidgetClass,
frame26,
args,
argcnt);

XtManageChild(radioBox1);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR( "All", XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNx, 3); argcnt++;
XtSetArg(args[argcnt], XmNy, 3); argcnt++;
toggleButton4 = XtCreateWidget("toggleButton4",
xmToggleButtonWidgetClass,
radioBox1,
args,
argcnt);
XmStringFree( xmstr[0] );

XtAddCallback(toggleButton4, XmNvalueChangedCallback, w2_all_val_changed_callback, (caddr_t)globals);
XtManageChild(toggleButton4);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR( "Loc", XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNx, 45); argcnt++;
XtSetArg(args[argcnt], XmNy, 3); argcnt++;
toggleButton5 = XtCreateWidget("toggleButton5",
xmToggleButtonWidgetClass,
radioBox1,
args,
argcnt);
XmStringFree( xmstr[0] );

XtAddCallback(toggleButton5, XmNvalueChangedCallback, w2_loc_val_changed_callback, (caddr_t)globals);
XtManageChild(toggleButton5);

argcnt = 0;
XtSetArg(args[argcnt], XmNlabelString, (xmstr[0]=XmStringCreateLtoR( "Type:"., XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, False); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 47); argcnt++;
XtSetArg(args[argcnt], XmNheight, 17); argcnt++;
label8 = XtCreateWidget("label8",
xmLabelWidgetClass,
frame25,
args,
argcnt);
XmStringFree( xmstr[0] );

XtManageChild(label8);

argcnt = 0;
/* XtSetArg(args[argcnt], XmNvalue, "type_int"); argcnt++;*/
XtSetArg(args[argcnt], XmNmarginHeight, (short)1); argcnt++;
XtSetArg(args[argcnt], XmNeditable, False); argcnt++;
XtSetArg(args[argcnt], XmNcursorPositionVisible, False); argcnt++;
XtSetArg(args[argcnt], XmNfontList, FONT_LIST[parent, \
procedure creation(text5, caddr_t A (globals->w2ids.text1), (caddr_t) 0);
XtManageChild(text5);

argcnt = 0;
XtSetArg(args[argcnt], XmLabelString, (xstr[0])=XmStringCreateLtoR("Accessed in.", XmSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argvcnt++;
XtSetArg(args[argcnt], XmNy, 2); argvcnt++;
label9 = XtCreateWidget("label9", xmLabelWidgetClass, frame23, args, argcnt);
XmStringFree( xstr[0] );

XtManageChild(label9);

argcnt = 0;
XtSetArg(args[argcnt], XmLabelString, (xstr[0])=XmStringCreateLtoR("Find", XmSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, True); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argvcnt++;
XtSetArg(args[argcnt], XmNy, 2); argvcnt++;
pushButton8 = XtCreateWidget("pushButton8", xmPushButtonWidgetClass, frame22, args, argcnt);
XmStringFree( xstr[0] );

XtAddCallback(pushButton8, XmNactivateCallback, find_list_item_callback, (caddr_t)0);
XtManageChild(pushButton8);

argcnt = 0;
XtSetArg(args[argcnt], XmLabelString, (xstr[0])=XmStringCreateLtoR("Exit", XmSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, False); argvcnt++;
XtSetArg(args[argcnt], XmNx, 2); argvcnt++;
XtSetArg(args[argcnt], XmNy, 2); argvcnt++;
XtSetArg(args[argcnt], XmNwidth, 64); argvcnt++;
XtSetArg(args[argcnt], XmNheight, 17); argvcnt++;
pushButton9 = XtCreateWidget("pushButton9", xmPushButtonWidgetClass, frame21, args, argcnt);
XmStringFree(xmstr[0]);

/* XAddCallback(pushButton9, XmNactivateCallback, exit_active_callback, (caddr_t)0); */
XtManageChild(pushButton9);

return( retval );
}

/* Source code widget */
/* --------------------------------------------------------------- */
Widget Createform2(parent, globals)
Widget parent;
Globals *globals;
{
Arg args[512];
imt argcnt;
Widget retval;
XmString xstr[32];
XmString *xstrTable[8];
Widget form2;
Widget frame20;
Widget label7;
Widget frame19;
Widget text3;
Widget frame18;
Widget menuBar;
Widget menuShell;
Widget pulldownMenu;
Widget pushButton5;
Widget cascadeButton;
Widget menuShell1;
Widget pulldownMenu1;
Widget pushButton6;
Widget cascadeButton1;
Widget frame17;
Widget scrolledWindow5;
Widget text4;
Widget frame16;
Widget pushButton7;
XmRegisterConverters();

argcnt = 0;
XtSetArg(args[argcnt], XmNx, 0); argcnt++;
XtSetArg(args[argcnt], XmNy, 0); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 630); argcnt++;
XtSetArg(args[argcnt], XmNheight, 919); argcnt++;
form2 = XtCreateWidget("form2", xmFormWidgetClass, parent, args, argcnt);

retval = form2;

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (Int)96); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (Int)99); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (Int)2); argcnt++;
XmSetArg(args[argcnt], XmNrightPosition, (int)18); argcnt++;
XmSetArg(args[argcnt], XmNX, 13); argcnt++;
XmSetArg(args[argcnt], XmNY, 876); argcnt++;
XmSetArg(args[argcnt], XmNwidth, 100); argcnt++;
XmSetArg(args[argcnt], XmNheight, 28); argcnt++;
frame16 = XCreateWidget("frame16",
xmFrameWidgetClass,
form2,
args,
argcnt);

XtManageChild(frame16);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)5); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)95); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)2); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)98); argcnt++;
XtSetArg(args[argcnt], XmNX, 13); argcnt++;
XtSetArg(args[argcnt], XmNY, 46); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 604); argcnt++;
XtSetArg(args[argcnt], XmNheight, 821); argcnt++;
frame17 = XCreateWidget("frame17",
xmFrameWidgetClass,
form2,
args,
argcnt);

XtManageChild(frame17);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)96); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)99); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)77); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)94); argcnt++;
XtSetArg(args[argcnt], XmNX, 485); argcnt++;
XtSetArg(args[argcnt], XmNY, 676); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 107); argcnt++;
XtSetArg(args[argcnt], XmNheight, 28); argcnt++;
frame18 = XCreateWidget("frame18",
xmFrameWidgetClass,
form2,
args,
argcnt);

XtManageChild(frame18);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)1); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)4); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)23); argcnt++;

155
XtSetArg(args[argcnt], XmNrightPosition, (int)70); argcnt++;
XtSetArg(args[argcnt], XmNx, 145); argcnt++;
XtSetArg(args[argcnt], XmNy, 9); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 296); argcnt++;
XtSetArg(args[argcnt], XmNheight, 28); argcnt++;
frame19 = XtCreateWidget("frame19",
xmlFrameWidgetClass,
form2,
args,
argcnt);

XtManageChild(frame19);

argcnt = 0;
XtSetArg(args[argcnt], XmNtopAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNbottomAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNleftAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNrightAttachment, XmATTACH_POSITION); argcnt++;
XtSetArg(args[argcnt], XmNtopPosition, (int)11); argcnt++;
XtSetArg(args[argcnt], XmNbottomPosition, (int)4); argcnt++;
XtSetArg(args[argcnt], XmNleftPosition, (int)5); argcnt++;
XtSetArg(args[argcnt], XmNrightPosition, (int)20); argcnt++;
XtSetArg(args[argcnt], XmNx, 32); argcnt++;
XtSetArg(args[argcnt], XmNy, 9); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 94); argcnt++;
XtSetArg(args[argcnt], XmNheight, 28); argcnt++;
frame20 = XtCreateWidget("frame20",
xmlFrameWidgetClass,
form2,
args,
argcnt);

XtManageChild(frame20);

argcnt = 0;
XtSetArg(args[argcnt], XmNLabelString, (xmstr[0]=XmStringCreateLtoR("Module"),
"STRING_DEFAULT_CHARSET")); argcnt++;
XtSetArg(args[argcnt], XmNrecomputeSize, False); argcnt++;
XtSetArg(args[argcnt], XmNx, 2); argcnt++;
XtSetArg(args[argcnt], XmNy, 2); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 90); argcnt++;
XtSetArg(args[argcnt], XmNheight, 24); argcnt++;
label7 = XtCreateWidget("label7",
xmlLabelWidgetClass,
frame20,
args,
argcnt);
XmStringFree( xmstr[0] );

XtManageChild(label7);

argcnt = 0;
XtSetArg(args[argcnt], XmNsashowThickness, (short)0); argcnt++;
/* XtSetArg(args[argcnt], XmNvalue, "initial\path"); argcnt++; */
XtSetArg(args[argcnt], XmNmarginHeight, (short)1); argcnt++;
/* XtSetArg(args[argcnt], XmNmarginWidth, (short)7); argcnt++; */
XtSetArg(args[argcnt], XmNfontList, FONT_LIST(parent:\n"-misc-fixed-medium+normal--10-100-75-75-c-60-iso8859-1")); argcnt++;
XtSetArg(args[argcnt], XmNeditable, False); argcnt++;
XtSetArg(args[argcnt], XmNcolumns, (short)20); argcnt++;
XtSetArg(args[argcnt], XmNrows, (short)1); argcnt++;
XtSetArg(args[argcnt], XmNresizeHeight, True); argcnt++;
argcnt = 0;
XtSetArg(args[argcnt], XmNwidth, 1); argcnt++;
XtSetArg(args[argcnt], XmNheight, 1); argcnt++;
menuShell = XtCreatePopupShell("menuShell",
                    xmMenuShellWidgetClass,
                    menuBar,
                    args,
                    argcnt);

XtManageChild(menuShell);

argcnt = 0;
XtSetArg(args[argcnt], XmNx, 0); argcnt++;
XtSetArg(args[argcnt], XmNlY, 0); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 42); argcnt++;
XtSetArg(args[argcnt], XmNheight, 31); argcnt++;
XtSetArg(args[argcnt], XmNrowColumnType, XmMENU_PULLDOWN); argcnt++;
        pulldownMenu = XtCreateWidget("pulldownMenu",
                    xmRowColumnTypeWidgetClass,
                    menuShell,
                    args,
                    argcnt);

argcnt = 0;
XtSetArg(args[argcnt], XmNalignment, XmALIGNMENT_BEGINNING); argcnt++;
XtSetArg(args[argcnt], XmNlabelString, (xstr)[0]=XmStringCreateToR(  "Load",
                    XmSTRING_DEFAULT_CHARSET)); argcnt++;
pushButton5 = XtCreateWidget("pushButton5",
                    xmPushButtonWidgetClass,
                    pulldownMenu,
                    args,
                    argcnt);
XmStringFree( xstr[0] );

XtAddCallback(pushButton5, XmNactivateCallback, w4_load_callback, (caddr_t)globals);
XtManageChild(pushButton5);

argcnt = 0;
XtSetArg(args[argcnt], XmNalignment, XmALIGNMENT_CENTER); argcnt++;
XtSetArg(args[argcnt], XmNmarginHeight, (short)0); argcnt++;
XtSetArg(args[argcnt], XmNlabelString, (xstr)[0]=XmStringCreateToR(  "Init",
                    XmSTRING_DEFAULT_CHARSET)); argcnt++;
XtSetArg(args[argcnt], XmNx, 12); argcnt++;
XtSetArg(args[argcnt], XmNlY, 2); argcnt++;
XtSetArg(args[argcnt], XmNwidth, 40); argcnt++;
XtSetArg(args[argcnt], XmNheight, 20); argcnt++;
XtSetArg(args[argcnt], XmNsubMenuId, pulldownMenu); argcnt++;
cascadeButton = XtCreateWidget("cascadeButton",
                    xmCascadeButtonWidgetClass,
                    menuBar,
                    args,
                    argcnt);
XmStringFree( xstr[0] );

XtManageChild(cascadeButton);
argcnt = 0;
XtSetArg(argcnt, XmNwidth, 1); argcnt++;
XtSetArg(argcnt, XmNheight, 1); argcnt++;
menuShell1 = XtCreatePopupShell("menuShell1",
XmMenuShellWidgetClass,
menuBar,
args,
argcnt);
XtManageChild(menuShell1);

argcnt = 0;
XtSetArg(argcnt, XmNx, 0); argcnt++;
XtSetArg(argcnt, XmNy, 0); argcnt++;
XtSetArg(argcnt, XmNwidth, 42); argcnt++;
XtSetArg(argcnt, XmNheight, 31); argcnt++;
XtSetArg(argcnt, XmNrowColumnType, XmMENU_PULLDOWN); argcnt++;
pulldownMenu1 = XtCreateWidget("pulldownMenu1",
XmRowColumnWidgetClass,
menuShell1,
args,
argcnt);

argcnt = 0;
XtSetArg(argcnt, XmNalignment, XmALIGNMENT_BEGINNING); argcnt++;
XtSetArg(argcnt, XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Exit",
XmSTRING_DEFAULT_CHARSET))); argcnt++;
pushButton6 = XtCreateWidget("pushButton6",
XmPushButtonWidgetClass,
pulldownMenu1,
args,
argcnt);
XmStringFree( xmstr[0] );

XtAddCallback(pushButton6, XmNactivateCallback, w4_exit_callback, (caddr_t)0);
XtManageChild(pushButton6);

argcnt = 0;
XtSetArg(argcnt, XmNalignment, XmALIGNMENT_CENTER); argcnt++;
XtSetArg(argcnt, XmNmarginHeight, (short)(0)); argcnt++;
XtSetArg(argcnt, XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Exit",
XmSTRING_DEFAULT_CHARSET))); argcnt++;
XtSetArg(argcnt, XmNx, 60); argcnt++;
XtSetArg(argcnt, XmNy, 2); argcnt++;
XtSetArg(argcnt, XmNwidth, 40); argcnt++;
XtSetArg(argcnt, XmNheight, 20); argcnt++;
XtSetArg(argcnt, XmNsubMenuId, pulldownMenu1); argcnt++;
cascadeButton1 = XtCreateWidget("cascadeButton1",
XmCascadeButtonDownWidgetClass,
menuBar,
args,
argcnt);
XmStringFree( xmstr[0] );

XtManageChild(cascadeButton1);

argcnt = 0;
XtSetArg(argcnt, XmNlabelString, (xmstr[0]=XmStringCreateLtoR("Find",

159
XmSTRING_DEFAULT_CHARSET()); argc++;
XtSetArg(args[argc], XmNrecomputeSize, False); argc++;
XtSetArg(args[argc], XmNx, 2); argc++;
XtSetArg(args[argc], XmNy, 2); argc++;
XtSetArg(args[argc], XmNwidth, 96); argc++;
XtSetArg(args[argc], XmNheight, 24); argc++;
pushButton7 = XtCreateWidget("pushButton7",
xmPushButtonWidgetClass,
frame16,
args,
argc);
XmStringFree( xstr[0] );

XtAddCallback(pushButton7, XmNactivateCallback, w4_find_callback, (caddr_t)0);
XtManageChild(pushButton7);

return( retval );
}
/* Thoerry Le Gal May 13, 1992 */
/* callbacks.c */
/* Contain the callbacks and the procedures for the ISCA */
/* */

#include <Xm/Xm.h>
#include "isca_struct.h"
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/List.h>
#include <stdio.h>
#include <sys/stat.h>
#include <sys/types.h>

#define CURRENT_SELECTED_MODULE "/current_selected_module"

/* Function declaration */
void w1_all_val_changed_callback();
void w1_priv_val_changed_callback();
void w1_lib_val_changed_callback();
void w1_io_val_changed_callback();
void w1_get_selected_list_item_callback();
void w1_load_list();
void w1_clean_widget();
void w1_load_retrieved_parameters_for_module();
void w1_load_retrieved_global_var_for_module();
void w1_load_retrieved_calls_for_module();
void w1_load_retrieved_called_by_for_module();
void w1_update_module_info();
void add_to_list();
void clear_list();
void clear_text();
void creation();
int list_size();
void find_list_item_callback();
void sort_list_callback();
int compare();
void w2_all_val_changed_callback();
void w2_loc_val_changed_callback();
void w2_get_selected_list_item_callback();
void w2_load_list();
void w2_update_list7();
int load_file_in_text_widget();
void w4_load_callback();
void w4_exit_callback();
void w4_find_callback();

/* */
/* Module list widget */
/* */

/* */
/* Is called when the value of toggle button "All" changed */

void w1_all_val_changed_callback(w,client, call)
Widget w;
caddr_t client;
caddr_t call;
{ /*
XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
Globals *globals=(Globals*)client;
int state;
int kind=1; /* All the modules will be loaded */

/* Get the state of the toggle button (1:True, on, enforce) (0:False, off, not enforce) */
state = XmToggleButtonGetState(w);

if (state == 1) {
clear_list(globals->w1ids.list1);
w1_load_list(globals->w1ids.list1,kind);
}
else {
clear_list(globals->w1ids.list1);
}

/* Clean all the widgets */
w1_clean_widget((caddr_t)globals);

/*-----------------------------------------------*/
/* Is called when the value of toggle button "Priv" changed */
/*-----------------------------------------------*/
void w1_priv_val_changed_callback(w,client,call)
Widget w;
caddr_t client;
caddr_t call;
{
XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
Globals *globals=(Globals*)client;
int state;
int kind=0; /* The private modules will be loaded */

/* Get the state of the toggle button (1:True, on, enforce) (0:False, off, not enforce) */
state = XmToggleButtonGetState(w);

if (state == 1) {
clear_list(globals->w1ids.list1);
w1_load_list(globals->w1ids.list1,kind);
}
else {
clear_list(globals->w1ids.list1);
}

/* Clean all the widgets */
w1_clean_widget((caddr_t)globals);

/*-----------------------------------------------*/
/* Is called when the value of toggle button "Lib" changed */
/*-----------------------------------------------*/
void w1_lib_val_changed_callback(w,client,call)
Widget w;
caddr_t client;
caddr_t call;
{
XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
Globals *globals=(Globals*)client;
int state;
int kind=2; /* The library modules will be loaded */

/* Get the state of the toggle button (1:True, on, enforce) (0:False, off, non enforce) */
state = XmToggleButtonGetState(w);

if (state == 1) {
clear_list(globals->w1ds.list1);
w1_load_list(globals->w1ds.list1.kind);
}
else {
clear_list(globals->w1ds.list1);
}

/* Clean all the widgets */
w1_clean_widget((caddr_t)globals);

/*
* is called when the value of toggle button 'i/o' changed
*/

void w1_io_val_changed_callback(w, client, call)
{ 
    Widget w;
    caddr_t client;
    caddr_t call;
    XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
    Globals *globals=(Globals*)client;
    int state;
    int kind=1; /* The i/o functions will be loaded */

    /* Get the state of the toggle button (1:True, on, enforce) (0:False, off, non enforce) */
    state = XmToggleButtonGetState(w);

    if (state == 1) {
clear_list(globals->w1ds.list1);
w1_load_list(globals->w1ds.list1.kind);
    }
else {
clear_list(globals->w1ds.list1);
    }

    /* Clean all the widgets */
w1_clean_widget((caddr_t)globals);
}

/*
* get the current selected item in the list
*/

void w1_get_selected_list_item_callback(w, client, list_data)
{ 
    Widget w;
    caddr_t client;
    XmListCallbackStruct *list_data;
    char *string;
    FILE *fp;
    XmStringGetLtoR(list_data->item, XmSTRING_DEFAULT_CHARSET, &string);
    /* print( "item was %s, item # %d\n", string, list_data->item_position ); */
}
/* Save the selected module value in a flat file */
if ((fp = fopen(CURRENT_SELECTED_MODULE, "w")) == NULL) {
    perror("error opening data file");
}
fprintf(fp, "%s\n", "current_selected module in widget1");
fprintf(fp, "%s\n", string);
fclose(fp);

w1_update_module_info(client, string);
 XtFree(string);
}

/* remove the information from the widget */

/* Processes worth for any widget */

/* Set the id for a widget at the creation */

void creation(w, client, call)
Widget w;
  caddr_t client;
  caddr_t call;
  { 
    XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
    Widget *setme=(Widget*)client;
    *setme = w;
  }

/* Add a text string to a list widget at a given position in a list */

void add_to_list(w, string, position)
Widget w;
  char *string;
  int position;
  { 
    
  }
XmString motif_string;
motif_string = XmStringCreateLtoR(string_XmSTRING_DEFAULT_CHARSET);
XmListAddItem(w, motif_string, position);
XmStringFree(motif_string);

/* Get the size of a list */
int list_size(w)
Widget w;
{
    int size;
    Arg args[10];

    XtSetArg(args[0], XmNitemCount, &size);
    XtGetValues(w, args, 1);
    return(size);
}

/* Clear a list */
void clear_list(w)
Widget w;
{
    int max;

    max = list_size(w);
    for (i=2;i<=max;i++) {
        XmListDeletePos(w, 0); /* delete the first item */
    }
    XmListDeletePos(w, 1); /* delete the first item */
}

/* Clear a text widget */
void clear_text(w)
Widget w;
{
    int status;

    /* Load the file into the text widget */
    status = load_file_in_text_widget(w, "/u/legals/bsc/1ac/blank_file.c");

    if (status == False) {
        printf("The file could not be loaded\n");
    }
}

/* Find an item in a list */
void find_list_item_callback(w, client, call)
Widget w;
caddr_t client;
caddr_t call;

{  XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
}

/* Sort the items in a list */
/* */
void sort_list_callback(w, client, call)
Widget w;
caddr_t client;
caddr_t call;
{  XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
  XtCloseDisplay(XtDisplay(w));
  exit(0);
}

/* Compare two C strings */
/* */
int compare(str1,str2)
char *str1;
char *str2;
{
  int i=0, answer;
  while (str1[i] == str2[i] & & str1[i] != '0' & & str2[i] != '0')
    ++i;
  if (str1[i] == '0' & & str2[i] == '0')
    answer = True; /* Strings equal */
  else
    answer = False; /* Strings not equal */
  return (answer);
}

/* Module detail widget */
/* */
/* */
/* is called when the value of the toggle button "All" changes */
/* */
void w2_all_val_changed_callback(w,client, call)
Widget w;
caddr_t client;
caddr_t call;
{  XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
  Globals *globals=(Globals*)client;
  int state,flag;
  char *name;

  /* Get the state of the toggle button (1:True, on, enforce) (0:False, off, non enforce) */
  state = XmToggleButtonGetState(w);
  if (state == 1) {

  }
}
clear_list(globals->w2ids.list1);
flag = 1;
w2_load_list(globals->w2ids.list1,flag);
clear_list(globals->w2ids.list2);
}
else {
clear_list(globals->w2ids.list1);
}

/* Clear the text fields */
name = " ";
XmTextSetString(globals->w2ids.text1, name);

/* is called when the value of the toggle button "Loc" changes */

/* ...

void w2_loc_val_changed_callback(w,client,call)
Widget w;
caddr_t client;
caddr_t call;
{
XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
Globals *globals=(Globals*)client;
int state,flag;
char *name;

/* Get the state of the toggle button (1:True,on,enable) (0:False,off,disable) */
state = XmToggleButtonGetState(w);

if (state == 1) {
clear_list(globals->w2ids.list1);
flag = 0;
w2_load_list(globals->w2ids.list1,flag);
clear_list(globals->w2ids.list2);
}
else {
clear_list(globals->w2ids.list1);
}

/* Clear the text fields */
name = " ";
XmTextSetString(globals->w2ids.text1, name);
/* XtFree(selected_list_item); */

/* ...

void w2_get_selected_list_item_callback(w,client,list_data)
Widget w;
caddr_t client;
XmListCallbackStruct *list_data;
{
char *string;

XmStringGetStr(list_data->item, XmSTRING_DEFAULT_CHARSET, &string);
/* printf( "item was %s, item # %d\n", string, list_data->item_position ); */
w2_update_list7(client,string);
XtFree( string );
}

/* Source code widget */

/* Is called when the "Load" option is selected from the menu */

void w4_load_callback(w,client,call)
Widget w;
caddr_t client;
caddr_t call;
{
    XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
    Globals *globals=(Globals*)client;
    char *command;
    int returned_val;

    returned_val = load_file_in_text_widget(globals->w4ids.text2, "/u/legal/bx/isca/dm/callbacks.c.c");
    if (returned_val == False) {
        printf("The file could not be loaded\n");
    } /*
    command = "/u/legal/analyzer/dbm/list_modules";
    system(command);
}

/* Exit the application */

void w4_exit_callback(w, client, call)
Widget w;
caddr_t client;
caddr_t call;
{
    XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;

    XtCloseDisplay(XtDisplay(w));
    exit(0);
}

/* Find a word in text */

void w4_find_callback(w, client, call)
Widget w;
caddr_t client;
caddr_t call;
{
    XmAnyCallbackStruct *acs=(XmAnyCallbackStruct*)call;
}

/*****************************/
/* Fills a multiline text widget with the content of an ASCII file */

int load_file_in_text_widget( widget, filename )

    Widget  widget;
    char    *filename[];

    {  
        FILE   *fp;
        struct stat    file_info;
        char    *buffer;
        long    bytes_read;

        fp = fopen( filename, "r" );
        if ( fp == NULL ) {
            return ( False );
        }

        /* get file size */
        if ( stat( filename, &file_info ) != 0 ) {
            fclose( fp );
            return( False );
        }

        buffer = (char *) XtMalloc( file_info.st_size + 5 );

        if ( buffer == (char *) NULL )
            {  
                fclose( fp );
                return( False );
            }

        bytes_read = fread( buffer, 1, file_info.st_size, fp );
        fclose( fp );

        buffer[ file_info.st_size - 1 ] = '\0';

        if ( bytes_read < file_info.st_size )
            {  
                XtFree( buffer );
                return( False );
            }
        XmTextSetString( widget, buffer ); /* place the text widget */
        XtFree( buffer );
        return( True );
    }
/* load_processes.c */
/* Thierry Le Gal       June 10, 1992 */
/* Functions used to extract some data from the data structure and update the ISCA display manager */
/* Most of these functions are called in /callsbacks-c.c */
/* */

#include <Xm/Xm.h>
#include "isca_struct.h"
#include <X11/Intrinsic.h>
#include <X11/StringDefs.h>
#include <Xm/List.h>
#include <stdio.h>
#include <sys/file.h>
#include <sys/stat.h>
#include <sys/types.h>
#include "data_struct.h" /* Contains the data structure */

#define RAW_DATA     "./parser/demo/raw.data"
#define RAW_DATA     "/u/leal/ibm/parser/these/raw.data.appl2"
#define KEY_FILE     "/key_file"
#define LINE_LEN 120
#define CURRENT_SELECTED_MODULE "/current_selected_module"
#define FILTER_FILE  "/filter_file"

/* Function definition */
void load_data_struct ();
void init_dbmdata();
void process_line1();
void process_line2();
void process_parameters();
void process_global_var();
void process_invoked_fct();
void process_invoked_lib();
void process_invoked_id();
void process_caller();
char *nextword();
char *restofline();
void w1_load_list();
void w1_update_module_info();
void w2_load_list();
void w2_update_list7();
char *filter();

/* */
void load_data_struct ()
{

    struct dbmdata_s *dbmptr, *head_dbmptr; /* pointer to the data structure */
    FILE *fp;
    /* file from which to read raw data */
    FILE *fp_key;
    char *inline;


    /* Open the raw data file */
    if ((fp = fopen(RAW_DATA, "r")) == NULL) {
        perror("error opening data file");

    }
head_dbmptr = (struct dbmdata_s *) malloc (1, sizeof(struct dbmdata_s));
if (head_dbmptr == NULL) {
  perror("not enough space on device; could not allocate memory");
}

/* Initialize the values in the dbmdata_s */
init_dbmdata(head_dbmptr);

inline = (char *) malloc(LINE_LEN);
if (inline == (char *) NULL) {
  perror("not enough space on device; could not allocate memory");
}

/* Get and load the first dbmdata element */
if (fgets(inline, LINE_LEN, fp) == NULL) {
  perror("error, raw data is empty");
}
process_line1(inline, head_dbmptr);
process_line2(fp, head_dbmptr);
process_parameters(fp, head_dbmptr);
process_global_var(fp, head_dbmptr);
process_invoked_fct(fp, head_dbmptr);
process_invoked_lib(fp, head_dbmptr);
process_invoked_ic(fp, head_dbmptr);
process_caller(fp, head_dbmptr);
head_dbmptr->next = NULL;

/* Scan the data from the file and load the info into the data structure */
while (fgets(inline, LINE_LEN, fp) != NULL) {
  dbmptr = (struct dbmdata_s *) malloc (1, sizeof(struct dbmdata_s));
  if (dbmptr == NULL) {
    perror("not enough space on device; could not allocate memory");
  }
  process_line1(inline, dbmptr);
  process_line2(fp, dbmptr);
  process_parameters(fp, dbmptr);
  process_global_var(fp, dbmptr);
  process_invoked_fct(fp, dbmptr);
  process_invoked_lib(fp, dbmptr);
  process_invoked_ic(fp, dbmptr);
  process_caller(fp, dbmptr);
  dbmptr->next = head_dbmptr;
  head_dbmptr = dbmptr;
}

/* Save the address of the pointer to the data structure in a flat file */
if ((fp_key = fopen(KEY_FILE,"w")) == NULL) {
  perror("error opening data file");
}
fprintf(fp_key,"%s\n","address of the pointer to the data structure");
fprintf(fp_key,"%d",head_dbmptr);
close(fp_key);
free(inline);

/* init_dbmdata */
/* Initialize the values in the dbmptr structure. */
void init_dbmdata(dbmptr)
struct dbmdata_s *dbmptr;
{  
  strcpy(dbmptn->fct_name,"");  
  strcpy(dbmptn->type,"");  
  dbmptn->kind = 0;  
  strcpy(dbmptn->filename,"");  
  dbmptn->line_num = 0;  
  dbmptn->param_count = 0;  
  dbmptn->locglobal_count = 0;  
  dbmptn->call_fct_count = 0;  
  dbmptn->call_io_count = 0;  
  dbmptn->call_lib_count = 0;  
  dbmptn->calledby_count = 0;  
}  

/*...........................................................................*/  
/* process_line1 */  
/* line1 format: <module name> <kind> <type> */  
/* example: main 0 int */  
/* Parse the values on line1 into the appropriate variables in the */  
/* dptr structure */  
/*........................................................................*/

void process_line1(aline,dptr)  
char *aline;  
struct dbmndata_s *dptr;  
{
  int offset=0;  
  char *tmptr;

  /* Remove final line feed */  
  aline[strlen(aline)-1] = '0';  
  strcpy(dpstr->fct_name,nextword(aline,&offset));

  /* Kind is an integer in the dbm data data structure, read as a string and convert */
  tmptr = (char *)malloc(IDENT_MAX);  
  strcpy(tmptr,nextword(aline,&offset));  
  sscanf(tmptr,"%d", &dpstr->kind);  
  free(tmptr);

  /* Last string or first line is the type of the module */
  strcpy(dpstr->type,restofline(aline,&offset));

  /* printf("%s %d %s\n",dpstr->fct_name, dpstr->kind, dpstr->type); */
}

/*...........................................................................*/  
/* process_line2 */  
/* line2 format: <file name where the module is> <line number> */  
/* example: /usr/include/dmnc.h 2239 */  
/* Parse the values on line2 into the appropriate variables in the dptr structure */

void process_line2(fp,dptr)  
FILE *fp;  
struct dbmndata_s *dpstr;  
{
  int offset=0;  
  char *tmptr, *line;

  line = (char *)malloc(LINE_LEN);

  /* Process line2... */
}
if (line == (char *)NULL) {
    perror("not enough space on device; could not allocate memory");
}

/* Get the line containing the module name */
fgets(line, LINE_LEN, fp);

/* Remove final line feed */
line[strlen(line) - 1] = '0';
strcpy(dp->filename, nextword(line, &offset));

/* linenumber is an integer in the data structure; read as a string and convert */
tmptr = (char *)malloc(IDENT_MAX);
strcpy(tmptr, nextword(line, &offset));
sscanf(tmptr, "%d", &dp->line_num);
free(tmptr);

/* printf("%s %d\n", dp->filename, dp->line_num); */
free(line);
}

/*----------------------------------------------------------------------------*/
/* process_parameters */
/* Format: <number of parameters> */
/* <param 1: <type> */
/* Read the number of parameters, parse them and store them into the appropriate variables in the structure */
/*----------------------------------------------------------------------------*/

void process_parameters(FILE *fp, dp)
{
    struct dbmdatum *dp;
    {
        int offset=0, i=0;
        struct param_s *head_paramptr, *paramptr;
        char *tmptr;
        char *line;

        line = (char *)malloc(LINE_LEN);
        if (line == (char *)NULL) {
            perror("not enough space on device; could not allocate memory");
        }

        /* Get the line containing the number of parameters */
        fgets(line, LINE_LEN, fp);

        /* Remove final line feed */
        line[strlen(line) - 1] = '0';

        /* number of parameters is an integer; read as a string and convert */
tmptr = (char *)malloc(IDENT_MAX);
        if (tmptr == (char *)NULL) {
            perror("not enough space on device; could not allocate memory");
        }
        strcpy(tmptr, nextword(line, &offset));
        sscanf(tmptr, "%d", &dp->param_count);

        /* printf("%d\n", dp->param_count); */
        free(tmptr);

        if(dp->param_count != 0) {

            /* Read the rest of the parameters */
            for (i=0; i<dp->param_count; i++) {
                line = (char *)malloc(LINE_LEN);
                if (line == (char *)NULL) {
                    perror("not enough space on device; could not allocate memory");
                }

                fgets(line, LINE_LEN, fp);
                line[strlen(line) - 1] = '0';

                paramptr = (param_s *)malloc(sizeof(param_s));
                if (paramptr == NULL) {
                    perror("not enough space on device; could not allocate memory");
                }

                /* Read the parameter number */
                tmptr = (char *)malloc(IDENT_MAX);
                if (tmptr == NULL) {
                    perror("not enough space on device; could not allocate memory");
                }

                strcpy(tmptr, nextword(line, &offset));
                sscanf(tmptr, "%d", &paramptr->param_num);

                /* Read the parameter type */
                tmptr = (char *)malloc(IDENT_MAX);
                if (tmptr == NULL) {
                    perror("not enough space on device; could not allocate memory");
                }

                strcpy(tmptr, nextword(line, &offset));
                sscanf(tmptr, "%s", paramptr->param_type);

                /* Read the parameter value */
                tmptr = (char *)malloc(IDENT_MAX);
                if (tmptr == NULL) {
                    perror("not enough space on device; could not allocate memory");
                }

                strcpy(tmptr, nextword(line, &offset));
                sscanf(tmptr, "%s", paramptr->param_value);

            }

            free(tmptr);
        }
    }
}

173
/* Creates the header of the parameters */
head_paramptr = (struct param_s *) calloc (1, sizeof(struct param_s));
if (paramptr == NULL) {
    perror("not enough space on device; could not allocate memory");
}
fgets(line,LINE_LEN,fp);
offset = 0;
line[strlen(line)-1] = "\0";
strcpy(head_paramptr->name,nextword(line,&offset));
strcpy(head_paramptr->type,restofline(line,&offset));
head_paramptr->next = NULL;
*/

/* Loop through the parameters, store them in the data structure */
while (i < dptr->param_count-1) {
    paramptr = (struct param_s *) calloc (1, sizeof(struct param_s));
    if (paramptr == NULL) {
        perror("not enough space on device; could not allocate memory");
    }
    fgets(line,LINE_LEN,fp);
    offset = 0;
    line[strlen(line)-1] = "\0";
    strcpy(paramptr->name,nextword(line,&offset));
    strcpy(paramptr->type,restofline(line,&offset));
    /*
    printf("%s\n",paramptr->name,paramptr->type);
    paramptr->next = head_paramptr;
    head_paramptr = paramptr;
    i++;
    */
}

/* Link the parameters list to the main structure */
dptr->ptr_param = head_paramptr;
}
else {
    dpdr->ptr_param = NULL;
}
free(line);

/*==========================================================================*/
/* process_global_var */
/* Format: <number of global variables accessed by the module> */
/* <global var1> <type> */
/* Read the number of global variables, parse them and store them into the appropriate variables in the structure */
/*==========================================================================*/

void process_global_var(fp,dptr)
FILE *fp;
struct dbmdata_s *dptr;
{
    int offset=0, i=0;
    struct logglobalvar_s *head_logglobalvarptr, *logglobalvarptr;
    char *line, *tmptr;
    line = (char *) malloc(LINE_LEN);
    if (line == (char *)NULL) {
        perror("not enough space on device; could not allocate memory");
    }

    /* Get the line containing the number of global variables */
    fgets(line,LINE_LEN,fp);
/* Remove final line feed */
line[strlen(line)-1] = '\0';

/* number of global variables is an integer, read as a string and convert */
tmptr = (char*)malloc(IDENT_MAX);
if (tmptr == (char*)NULL) {
    perror("not enough space on device; could not allocate memory");
}
strcpy(tmptr, nextword(line,&offset));
scanf(tmptr,"%d", &dptr->logeglobalvar_count);
/* printf("%d\n", dptr->logeglobalvar_count); */
free(tmptr);

if (dptr->logeglobalvar_count != 0) {

    /* Creates the header of the global variables */
    head_logeglobalvarptr = (struct logeglobalvar_s*) alloca (1, sizeof(struct logeglobalvar_s));
    if (head_logeglobalvarptr == NULL) {
        perror("not enough space on device; could not allocate memory");
    }
    fgets(line,LINE_LEN,fp);
    offset = 0;
    line[strlen(line)-1] = '\0';
    strcpy(head_logeglobalvarptr->name, nextword(line,&offset));
    strcpy(head_logeglobalvarptr->type, restofline(line,&offset));
    head_logeglobalvarptr->next = NULL;
    printf("%s %s\n", head_logeglobalvarptr->name, head_logeglobalvarptr->type); */

    /* Loop through the global variables, store them in the data structure */
    while (i < dptr->logeglobalvar_count-1) {
        logeglobalvarptr = (struct logeglobalvar_s*) alloca (1, sizeof(struct logeglobalvar_s));
        if (logeglobalvarptr == NULL) {
            perror("not enough space on device; could not allocate memory");
        }
        fgets(line,LINE_LEN,fp);
        offset = 0;
        line[strlen(line)-1] = '\0';
        strcpy(logeglobalvarptr->name, nextword(line,&offset));
        strcpy(logeglobalvarptr->type, restofline(line,&offset));
        /* printf("%s %s\n", logeglobalvarptr->name, logeglobalvarptr->type); */
        /* Insert the element in the list */
        logeglobalvarptr->next = head_logeglobalvarptr;
        head_logeglobalvarptr = logeglobalvarptr;
        i++;
    }

    /* Link the parameters list to the main structure */
    dptr->ptr_logeglobalvar = head_logeglobalvarptr;
}
else {
    dptr->ptr_logeglobalvar = NULL;
}
free(line);

/* process_invoked_fct */
/* Format: <number of functions invoked by the module> */
<invoked_fct1> /*
/* Read the number of invoked functions, parse them and store them into the appropriate variables in the structure */
/*............................................................................................................*/

void process_invoked_fct(p, dptr)
FILE *fp;
struct dbndata_s *dptr;
{
    int offset = 0, i = 0;
    struct call_fct_s *head_call_fctptr, *call_fctptr;
    char *line, *tmptr;

    line = (char *)malloc(LINE_LEN);
    if (line == (char *)NULL) {
        perror("not enough space on device; could not allocate memory");
    }

    /* Get the line containing the number of global variables */
    fgets(line, LINE_LEN, fp);

    /* Remove final line feed */
    line[strlen(line)-1] = '\0';

    /* number of invoked functions is an integer; read as a string and convert */
    tmptr = (char *)malloc(IDENT_MAX);
    if (tmptr == (char *)NULL) {
        perror("not enough space on device; could not allocate memory");
    }
    strcpy(tmprt, nextword(line, &offset));
    sscanf(tmprt, "%d", &dptr->call_fct_count);
    /* printf("%d\n", dptr->call_fct_count); */
    free(tmprt);

    if (dptr->call_fct_count != 0) {

        /* Creates the header of the invoked functions */
        head_call_fctptr = (struct call_fct_s *)calloc(1, sizeof(struct call_fct_s));
        if (head_call_fctptr == NULL) {
            perror("not enough space on device; could not allocate memory");
        }
        fgets(line, LINE_LEN, fp);
        offset = 0;
        line[strlen(line)-1] = '\0';
        strcpy(head_call_fctptr->name, nextword(line, &offset));
        head_call_fctptr->next = NULL;
        /* printf(" %s\n", head_call_fctptr->name); */

        /* Loop through the invoked functions, store them in the data structure */
        while (i < dptr->call_fct_count - 1) {
            call_fctptr = (struct call_fct_s *)calloc(1, sizeof(struct call_fct_s));
            if (call_fctptr == NULL) {
                perror("not enough space on device; could not allocate memory");
            }
            fgets(line, LINE_LEN, fp);
            offset = 0;
            line[strlen(line)-1] = '\0';
            strcpy(call_fctptr->name, nextword(line, &offset));
            /* printf(" %s \n", call_fctptr->name); */
            /* Insert the element in the list */
            call_fctptr->next = head_call_fctptr;
            head_call_fctptr = call_fctptr;
            i++;
        }
    }
}

176
Link the invoked functions list to the main structure:
    dptr->ptr_call_fct = head_call_fctptr;
}
else {
    dptr->ptr_call_fct = NULL;
}
free(line);

process_invoked_lib

Format: <number of regular library invoked by the module>

Read the number of invoked libraries, parse them and store them into the appropriate variables in the structure:

void process_invoked_lib(FILE *fp, struct dbmdata_s *dptr)
{
    int offset=0, i=0;
    struct call_lib_s *head_call_libptr, *call_libptr;
    char *line, *tmpr;

    line = malloc(Line_LEN);
    if (line == NULL) {
        perror("not enough space on device; could not allocate memory");
    }

    /* Get the line containing the number of regular libraries */
    fgets(line, Line_LEN, fp);

    /* Remove final line feed */
    line[strlen(line)-1] = '\0';

    /* number of invoked libraries is an integer; read as a string and convert */
    tmpr = malloc(IDENT_MAX);
    if (tmpr == NULL) {
        perror("not enough space on device; could not allocate memory");
    }
    strncpy(tmpr, nextword(line, &offset));
    sscanf(tmpr, "%d", &dptr->call_lib_count);

    /* print("%dn", dptr->call_lib_count); */
    free(tmpr);

    if (dptr->call_lib_count == 0) {

        /* Creates the header of the invoked libraries */
        head_call_libptr = malloc(sizeof(struct call_lib_s));
        if (head_call_libptr == NULL) {
            perror("not enough space on device; could not allocate memory");
        }
        fgets(line, Line_LEN, fp);
        offset = 0;
        line[strlen(line)-1] = '\0';
        strncpy(head_call_libptr->name, nextword(line, &offset));
        head_call_libptr->next = NULL;
        /* print(" %s\n", head_call_libptr->name); */
Loop through the invoked libraries, store them in the data structure */

while (i < dptr->call_lib_count - 1) {
    call_libptr = (struct call_lib_s *) calloc (1, sizeof(struct call_lib_s));
    if (call_libptr == NULL) {
       perror("not enough space on device; could not allocate memory");
    }
    fgets(line,LINE_LEN,fp);
    offset = 0;
    line[strlen(line)-1] = '\0';
    strcpy(call_libptr->name,nextword(line,&offset));

    printf(" %s\n",call_libptr->name);

    Insert the element in the list */
    call_libptr->next = head_call_libptr;
    head_call_libptr = call_libptr;
    i++;
}

" Link the invoked libraries list to the main structure */

dptr->ptr_call_lib = head_call_libptr;
}

else {
    dptr->ptr_call_lib = NULL;
}
free(line);

/.......................................................................*/
/* process_invoked_io */
/* Format: <number of io libraries invoked by the module> */
/* <invoked io1> */
/* Read the number of invoked io libraries, parse them, store them into the appropriate variables in the structure */
/* ............................................................................................................*/

void process_invoked_io(FILE *fp, dptr_t *dptr) {

    struct dbmdata_s *dptr;

    int offset=0, i=0;
    struct call_io_s *head_call_ioptr, *call_ioptr;
    char *line, *tmpstr;

    line = (char *)malloc(LINE_LEN);
    if (line == (char *) NULL) {
       perror("not enough space on device; could not allocate memory");
    }

    /* Get the line containing the number of io libraries */
    fgets(line,LINE_LEN,fp);

    /* Remove final line feed */
    line[strlen(line)-1] = '\0';

    /* number of invoked io libraries is an integer; read as a string and convert */
    tmpstr = (char *)malloc(IDENT_MAX);
    if (tmpstr == (char *) NULL) {
       perror("not enough space on device; could not allocate memory");
    }
    strcpy(tmpstr,nextword(line,&offset));
    sscanf(tmpstr,"%d", &dptr->call_io_count);
    printf("%dn", dptr->call_io_count);
    free(tmpstr);
}
if (dptr->call_io_count)
{
    /* Creates the header of the invoked libraries */
    head_call_ioptr = (struct call_io_s *) calloc (1, sizeof(struct call_io_s));
    if (head_call_ioptr == NULL) {
        perror("not enough space on device; could not allocate memory");
    }
    fgets(line,LINE_LEN,fp);
    offset = 0;
    line[strlen(line)-1] = '\0';
    strcpy(head_call_ioptr->name,nextword(line,&offset));
    head_call_ioptr->next = NULL;
    printf("%s\n",head_call_ioptr->name);
    /*
    Loop through the invoked io libraries, store them in the data structure */
    while (i < dptr->call_io_count-1) {
        call_ioptr = (struct call_io_s *) calloc (1, sizeof(struct call_io_s));
        if (call_ioptr == NULL) {
            perror("not enough space on device; could not allocate memory");
        }
        fgets(line,LINE_LEN,fp);
        offset = 0;
        line[strlen(line)-1] = '\0';
        strcpy(call_ioptr->name,nextword(line,&offset));
        printf("%s\n",call_ioptr->name);
        /* Insert the element in the list */
        call_ioptr->next = head_call_ioptr;
        head_call_ioptr = call_ioptr;
        i++;
    }
    /* Link the invoked io libraries list to the main structure */
    dptr->ptr_call_io = head_call_ioptr;
}
else {
    dptr->ptr_call_io = NULL;
}
free(line);

/*
*/

/* process_caller */
/* Format: number of modules calling the module */
/*
*/
/* Read the number of calling modules, parse them and store them into the appropriate variables in the structure */
void process_caller(FILE *fp, dptr)
FILE *fp;
struct dbmdata_s **dptr;
{
    int offset=0, i=0;
    struct calledby_s *head_calledbyptr, *calledbyptr;
    char *line, *tmptr;
    line = (char *) malloc(LINE_LEN);
    if (line == (char *) NULL) {
        perror("not enough space on device; could not allocate memory");
    }
}
/* Get the line containing the number of calling modules */
fgets(line,LINE_LEN,p);

/* Remove final line feed */
line[strlen(line)-1] = '\0';

/* number of calling modules is an integer; read as a string and convert */
tmpstr = (char *)malloc(IDENT_MAX);
if (tmpstr == (char *)NULL) {
    perror("not enough space on device; could not allocate memory");
}
strncpy(tmpstr,nextword(line,&offset));
sscanf(tmpstr,"%d", &dptr->calledby_count);
printf("%d\n", dptr->calledby_count);
free(tmpstr);

if (dptr->calledby_count != 0) {

    /* Creates the header of the calling modules */
    head_calledbyptr = (struct calledby_s *)malloc(1, sizeof(struct calledby_s));
    if (head_calledbyptr == NULL) {
        perror("not enough space on device; could not allocate memory");
    }
    fgets(line,LINE_LEN,p);
    offset = 0;
    line[strlen(line)-1] = '\0';
    strncpy(head_calledbyptr->name,nextword(line,&offset));
    head_calledbyptr->next = NULL;
    printf("%s \n", head_calledbyptr->name);

    /* Loop through the calling modules, store them in the data structure */
    while (i < dptr->calledby_count-1) {
        calledbyptr = (struct calledby_s *)malloc(1, sizeof(struct calledby_s));
        if (calledbyptr == NULL) { 
            perror("not enough space on device; could not allocate memory");
        }
        fgets(line,LINE_LEN,p);
        offset = 0;
        line[strlen(line)-1] = '\0';
        strncpy(calledbyptr->name,nextword(line,&offset));
        printf("%s \n", calledbyptr->name);
    }
    Insert the element in the list */
    calledbyptr->next = head_calledbyptr;
    head_calledbyptr = calledbyptr;
    i++;

    /* Link the invoked libraries list to the main structure */
    dptr->ptr_calledby = head_calledbyptr;
}
else {
    dptr->ptr_calledby = NULL;
}
free(line);

/* nextword */
/* return the next "word" (set of non-space characters) following the position of "offsetp" in the string "str" */
/*...........................................................................................................*/
char *nextword(str, offsetp)
    char *str;
    int *offsetp;
    {
        int i;
        char newstr[IDENT_MAX];

        /* Accumulate non-white characters in the newstr array. */
        /* Increment offset as you count through the string. */
        i = 0;
        while ((str[*offsetp + i] != ' ') &&
               (str[*offsetp + i] != 'n') &&
               (str[*offsetp + i] != 'O')) {
            newstr[i] = str[*offsetp + i];
            i++;
        }

        if (str[*offsetp + i] == ' )
            *offsetp = *offsetp + i + 1;
        else
            *offsetp = *offsetp + i;

        /* Terminate newstr with the nil character and return the string. */
        newstr[i] = 'O';
        return(newstr);
    }

/*---------------------------------------------------------------*/
/* restofline */
/* return the string containing the rest of the line of the "str" string following the position marked by "offsetp" */
/*---------------------------------------------------------------*/
char *restofline(str, offsetp)
    char *str;
    int *offsetp;
    {
        char newstr[LINELEN];
        strep(newstr, str + *offsetp);
        return(newstr);
    }

/*---------------------------------------------------------------*/
/* load the list of modules in widget1 */
/*---------------------------------------------------------------*/
void w1_load_list(client_kind)
    Widget client;
    int kind; /* -1 for all modules, 0 for private modules, 2 for regular library module, 1 for io library module */
    {
        FILE *fp, *fp;
        char *line, *tmptr, *module;
        int offset, address, exit_val;
        struct dbmdata_s *head_dbmpt, *current_dbmpt;
        struct call_s *current_call_libpt;
        struct call_s *current_call_iopr;
        char command[15];

        line = (char *) malloc(LINELEN);
        if (line == (char *) NULL) {
 perror("not enough space on device; could not allocate memory");
}

module = (char *) malloc(LINE_LEN);
if (module == (char *) NULL) {
 perror("not enough space on device; could not allocate memory");
}

load_data_struct();

/* Read the address of the pointer to the data structure from a flat file */
if ((fp_key = fopen(KEY_FILE,"r")) == NULL) {
 perror("error opening data file");
}
fgets(line,LINE_LEN,fp_key);
fgets(line,LINE_LEN,fp_key);
offset = 0;

tmpr = (char *) malloc(LENTH_MAX);
if (tmpr == (char *) NULL) {
 perror("not enough space on device; could not allocate memory");
}
strcpy(tmpr,nextword(line,&offset));
sscanf(tmpr,"%d"&address);
head_dbmptr = address;
free(tmpr);
free(line);
close(fp_key);

/* Create and flush the file FILTER_FILE */
if ((fp = fopen(FILTER_FILE,"w")) == NULL)
 perror("error opening data file");
fprintf(fp,%"s","t");
fclose(fp);

/* Loop through the main link list of the data structure (containing the module definition) selecting the right modules */
current_dbmptr = head_dbmptr;
exit_val = 0;
while (exit_val != 1) {
 if (kind == -1) {
 /* Load private modules, regular library modules, io library modules */

 if (current_dbmptr->kind != 3) { /* Get the private modules, skipping the global variables */
 module = filter(current_dbmptr->fct_name);
 if (strcmp(module, "") != 0)
 add_to_list(client, module, 0);
 }

 if (current_dbmptr->call_lib_count != 0) { /* Get the regular library modules */
 exit_val = 0;
current_call_libptr = current_dbmptr->ptr_call_lib;
while (exit_val != 1) {
 module = filter(current_call_libptr->name);
 if (strcmp(module, "") != 0)
 add_to_list(client, module, 0);
 if (current_call_libptr->next == NULL)
 exit_val = 1;
 else
 current_call_libptr = current_call_libptr->next;
 }
 }

 if (current_dbmptr->call_io_count != 0) { /* Get the io library modules */
 exit_val = 0;
current_call_ioptr = current_dbmptr->ptr_call_io;
}
while (exit_val != 1) {
    module = filter(current_call_ioptr->name);
    If (strcmp(module, "") != 0)
        add_to_list(client, module, 0);
    if (current_call_ioptr->next == NULL)
        exit_val = 1;
    else
        current_call_ioptr = current_call_ioptr->next;
}
/* End if kind == -1 */

if(kind == 0) /* Load private modules only */
    if (current_dbmptr->kind != 3) /* Get the private modules, skipping the global variables */
        module = filter(current_dbmptr->fct_name);
        if (strcmp(module, "") != 0)
            add_to_list(client, module, 0);
    }
    /* End if kind == 0 */

if(kind == 2) /* Load regular library modules only */
    if (current_dbmptr->kind != 3 && (current_dbmptr->call_lib_count != 0)) { /* Get the regular library modules */
        current_call_libptr = current_dbmptr->ptr_callLib;
        while (exit_val != 1) {
            module = filter(current_call_libptr->name);
            if (strcmp(module, "") != 0)
                add_to_list(client, module, 0);
            if (current_call_libptr->next == NULL)
                exit_val = 1;
            else
                current_call_libptr = current_call_libptr->next;
        }
    }
    /* End if kind == 2 */

if(kind == 1) /* Load io library modules only */
    if ((current_dbmptr->kind != 3 && (current_dbmptr->call_io_count != 0)) { /* Get the io library modules */
        current_call_ioptr = current_dbmptr->ptr_callIo;
        while (exit_val != 1) {
            module = filter(current_call_ioptr->name);
            if (strcmp(module, "") != 0)
                add_to_list(client, module, 0);
            if (current_call_ioptr->next == NULL)
                exit_val = 1;
            else
                current_call_ioptr = current_call_ioptr->next;
        }
    }
    /* End if == 2 */

/* Go to the next record in the main link list */
exit_val = 0;
if (current_dbmptr->next == NULL)
    exit_val = 1;
else
    current_dbmptr = current_dbmptr->next;
void w1_update_module_info(client, selected_list_item)

Widget client;
char *selected_list_item;
{

FILE *fp_key;
char *line, *tmptr, *param_description;
int offset, address, exit_val, found_module, status;
struct dbmdata_s *head_dbmpt, *current_dbmpt;
struct param_s *current_paramptr;
struct logglobalvar_s *current_logglobalvarptr;
struct call_fct_s *current_call_fctptr;
struct call_lib_s *current_call_libptr;
struct call_io_s *current_call_ioptr;
struct calledby_s *current_calledbyptr;
char name[2];

Globals *globals=(Globals*)client;

line = (char *) malloc(LINE_LEN);
if (line == (char *) NULL) {
    perror("not enough space on device; could not allocate memory");
}
param_description = (char *) malloc(LINE_LEN);
if (param_description == (char *) NULL) {
    perror("not enough space on device; could not allocate memory");
}

/* Clear all the objects on the widgets */
clear_list(globals->w3ids.list1);
clear_lst(globals->w3ids.list2);
clear_list(globals->w3ids.list3);
clear_list(globals->w3ids.list4);
clear_text(globals->w4ids.text2);

/* Clear the text fields */
strcpy(name, "");
XmTextSetString(globals->w3ids.text1, name);
XmTextSetString(globals->w3ids.text2, name);
XmTextSetString(globals->w3ids.text3, name);
XmTextSetString(globals->w4ids.text1, name);

load_data_struct();

/* Read the address of the pointer to the data structure from a flat file */
if ((fp_key = fopen(KEY_FILE, "r")) == NULL) {
    perror("error opening data file Key file");
}
gets(line, LINE_LEN, fp_key);
gets(line, LINE_LEN, fp_key);
offset = 0;
tmptr = (char *) malloc(IDENT_MAX);
if (tmptr == (char*) NULL) {
    perror("not enough space on device; could not allocate memory");
}
strcpy(tmptr.nextword(line, &offset));
sscanf(tmptr,"%d", &address);
head_dbmpt = address;
free(tmptr);
free(line);
fclose(fp_key);

/* Loop through the main link list of the data structure to find the module record */
current_dbmpt = head_dbmpt;
found_module = 0;
exit_val = 0;
while (found_module == 0 && exit_val != 1) {
    if (strcmp(current_dbmpt->fct_name, selected_list_item) == 0) {
        found_module = 1;
    }
    if (found_module == 0 && current_dbmpt->next != NULL) {
        current_dbmpt = current_dbmpt->next;
    }
    if (found_module == 0 && current_dbmpt->next == NULL) {
        if (strcmp(current_dbmpt->fct_name, selected_list_item) == 0) {
            found_module = 1;
        }
    }
    if (found_module == 0) {
        printf("%s is not stored in the data structure as a module definition record\n", selected_list_item);
        exit_val = 1;
    }
}
}

if (found_module == 1) {
    XmTextSetString(globals->w3ids.text1, current_dbmpt->fct_name);
    XmTextSetString(globals->w3ids.text2, current_dbmpt->type);
    XmTextSetString(globals->w3ids.text3, current_dbmpt->filename);
    XmTextSetString(globals->w4ids.text1, current_dbmpt->filename);

    /* Load the file into the text widget */
    /* status = load_file_in_text_widget(globals->w4ids.text2, current_dbmpt->filename); */
    if (status == False) {
        printf("Could not find the source code %s\n", current_dbmpt->filename);
    } /*

    /* status = load_file_in_text_widget(globals->w4ids.text2, "demo_display_source_code"); */
    status = load_file_in_text_widget(globals->w4ids.text2, "blank_file.c");
    if (status == False) {
        printf("Could not find the source code %s\n", current_dbmpt->filename);
    }

    /* Loop through the parameters link list */
    if (current_dbmpt->param_count != 0) {
        exit_val = 0;
        current_paramptr = current_dbmpt->ptr_param;
        while (exit_val != 1) {
            strcpy(param_description, current_paramptr->name);
            strcat(param_description, " ");
            strcat(param_description, current_paramptr->type);
            add_to_list(globals->w3ids.list1, param_description, 0);
        }
    }
}
if (current_paramptr->next == NULL) {
    exit_val = 1;
}
else {
    current_paramptr = current_paramptr->next;
}
}

/* Loop through the global variables link list */
if (current_dbm.ptr->localglobalvar_count != 0) {
    exit_val = 0;
    current_locglobalvarptr = current_dbm.ptr->ptr_locglobalvar;
    while (exit_val != 1) {
        add_to_list(globals->w3ids.list2, current_locglobalvarptr->name, 0);
        if (current_locglobalvarptr->next == NULL) {
            exit_val = 1;
        }
    }
    else {
        current_locglobalvarptr = current_locglobalvarptr->next;
    }
}

/* Loop through the invoked functions link list */
if (current_dbm.ptr->call_fct_count != 0) {
    exit_val = 0;
    current_call_fctptr = current_dbm.ptr->ptr_call_fct;
    while (exit_val != 1) {
        add_to_list(globals->w3ids.list3, current_call_fctptr->name, 0);
        if (current_call_fctptr->next == NULL) {
            exit_val = 1;
        }
    }
    else {
        current_call_fctptr = current_call_fctptr->next;
    }
}

/* Loop through the invoked libraries link list */
if (current_dbm.ptr->call_lib_count != 0) {
    exit_val = 0;
    current_call_libptr = current_dbm.ptr->ptr_call_lib;
    while (exit_val != 1) {
        add_to_list(globals->w3ids.list3, current_call_libptr->name, 0);
        if (current_call_libptr->next == NULL) {
            exit_val = 1;
        }
    }
    else {
        current_call_libptr = current_call_libptr->next;
    }
}

/* Loop through the invoked io link list */
if (current_dbm.ptr->call_io_count != 0) {
    exit_val = 0;
    current_call_ioptr = current_dbm.ptr->ptr_call_io;
    while (exit_val != 1) {
        add_to_list(globals->w3ids.list3, current_call_ioptr->name, 0);
        if (current_call_ioptr->next == NULL) {
            exit_val = 1;
        }
    }
    else {
        current_call_ioptr = current_call_ioptr->next;
    }
}
```c
}
else {
    current_call_ioptr = current_call_ioptr->next;
}
}

/* Loop through the called by link list */
if (current_dbmpt->calledby_count != 0) {
    exit_val = 0;
    current_calledbyptr = current_dbmpt->ptr_calledby;
    while (exit_val != 1) {
        add_to_list(globals->w3ids.list4,current_calledbyptr->name,0);
        if (current_calledbyptr->next == NULL) {
            exit_val = 1;
        } else {
            current_calledbyptr = current_calledbyptr->next;
        }
    }
}

/* Go to the next record in the main link list */
exit_val = 0;
if (current_dbmpt->next == NULL) {
    exit_val = 1;
} else {
    current_dbmpt = current_dbmpt->next;
}


/* load the list of global variables in widget2 */
/* load the list of global variables in widget2 */

void w2_load_list(client_flag)
Widget client;
int flag; /* 1 for all global var to be loaded, 0 for only local global var to be loaded */
{
    FILE *fp_key, *fp;
    char *line, *tmptr, *module;
    int offset, address, exit_val, found_module;
    struct dbm_ptr_t *head_dbmpt, *current_dbmpt;
    struct locglobalvar_t *current_locglobalvarptr;
    char last_selected_module[LINE_LEN];

    Globals *globals=(Globals*)client;

    line = (char *) malloc(LINE_LEN);
    if (line == (char *) NULL) 
        perror("not enough space on device; could not allocate memory");

    module = (char *) malloc(LINE_LEN);
    if (module == (char *) NULL) 
        perror("not enough space on device; could not allocate memory");

    load_data_struct();

    /* Read the address of the pointer to the data structure from a flat file */
    if ((fp_key = fopen(KEY_FILE,"r")) == NULL) {
```
perror("error opening data file Key file");
}
fgets(line,LINE_LEN,ip_key);
fgets(line,LINE_LEN,ip_key);
offset = 0;
tmptr = (char *)malloc(IDENT_MAX);
if (tmptr == (char *)NULL)
    perror("not enough space on device; could not allocate memory");
strcpy(tmptr,nextword(line,&offset));
sscanf(tmptr,"%d",&address);
head_dbm.ptr = address;
free(tmptr);
fclose(ip_key);

/* Create and flush the file FILTER_FILE */
if ((fp = fopen(FILTER_FILE,"w")) == NULL)
    perror("error opening data file");
 fprintf(fp,"%s:\n","");
fclose(fp);

/* Loop through the main link list of the data structure (containing the module definition) selecting the right modules */
current_dbm.ptr = head_dbm.ptr;

if (flag == 1) /* All the global var will be loaded */
ext Exit_val = 0;
while (exit_val != 1) {
    if (current_dbm.ptr->kind == 3) {
        module = filter(current_dbm.ptr->fct_name);
        if (strcmp(module, "") != 0)
            add_to_list(client, module, 0);
    }
    /* Go to the next record in the main link list */
    if (current_dbm.ptr->next == NULL)
        exit_val = 1;
    else
        current_dbm.ptr = current_dbm.ptr->next;
}

else if (flag == 0) /* Only the global var locally accessed by the last selected module will be loaded */
/* Read the value of the last selected module in widget */
if ((fp = fopen(CURRENT_SELECTED_MODULE,"r")) == NULL)
    perror("error opening data file");
fgets(line,LINE_LEN,fp);
fgets(line,LINE_LEN,fp);
offset = 0;
line[strlen(line)-1] = '\0';
strcpy(last_selected_module,nextword(line,&offset));
fclose(fp);

/* Loop through the main link list of the data structure to find the module record */
current_dbm.ptr = head_dbm.ptr;
found_module = 0;
exit_val = 0;
while (found_module == 0 && exit_val != 1) {
    if (strcmp(current_dbm.ptr->fct_name,last_selected_module) == 0)
        found_module = 1;
    if (found_module == 0 && current_dbm.ptr->next != NULL)
        current_dbm.ptr = current_dbm.ptr->next;
    if (found_module == 0 && current_dbm.ptr->next == NULL) {
        if (strcmp(current_dbm.ptr->fct_name,last_selected_module) == 0)
found_module = 1;
if (found_module == 0) {
    printf("%s is not stored in the data structure as a module definition record\n", last_selected_module);
    exit_val = 1;
}
}
}

if (found_module == 1) {
    /* Loop through the global variables link list */
    if (current_dbmaptr->locglobalvar_count != 0) {
        exit_val = 0;
        current_locglobalvarptr = current_dbmaptr->ptr_locglobalvar;
        while (exit_val != 1) {
            module = filter(current_locglobalvarptr->name);
            if (strcmp(module, "") != 0)
                add_to_lst(client, module, 0);
            if (current_locglobalvarptr->next == NULL)
                exit_val = 1;
            else
                current_locglobalvarptr = current_locglobalvarptr->next;
        }
    }
}

/* load the list of global variables in widget2 */
/* load the list of global variables in widget2 */
void w2_update_list7(client, list_item)
Widget client;
char *list_item;
{
    Globals *globals=(Globals*)client;

    FILE *fp_key;
    char *line, *ptr;
    int offset, address, exit_val, found_locglobalvar;
    struct dbmdata_s *head_dbmpr, *current_dbmpr;
    struct locglobalvar_s *current_locglobalvarptr;

    line = (char *)malloc(LINE_LEN);
    if (line == (char *)NULL)
        perror("not enough space on device; could not allocate memory");
    load_data_struct();

    /* Read the address of the pointer to the data structure from a flat file */
    if ((fp_key = fopen(KEY_FILE, "r")) == NULL) {
        perror("error opening data file Key file");
    }
    fgets(line, LINE_LEN, fp_key);
    fgets(line, LINE_LEN, fp_key);
    offset = 0;
    ptr = (char *)malloc(IDENT_MAX);
    if (ptr == (char *)NULL) {
        perror("not enough space on device; could not allocate memory");
    }
```c
    strcpy(tmpstr.nextword(line,&offset));
sscanf(tmpstr,"%d",&address);
head_dblmpt = address;
free(tmpstr);
tclose(fp_key);

/* Loop through the main link list of the data structure */
current_dblmpt = head_dblmpt;
exit_val = 0;
found_locglobalvar = 0;
while (exit_val != 1) {

    if (current_dblmpt->locglobalvar_count != 0) {
        exit_val = 0;
        current_locglobalvarptr = current_dblmpt->ptr_locglobalvar;
        while (exit_val != 1) { /* Loop through the global var list */
            if (strcmp(list_item,current_locglobalvarptr->name) == 0) {
                clear_list(globals->w2ids.list2);
                add_to_list(globals->w2ids.list2,current_dblmpt->filename,0);
                clear_text(globals->w2ids.text1);
                XmTextSetString(globals->w2ids.text1,current_locglobalvarptr->type);
                found_locglobalvar = 1;
            }
            if (current_locglobalvarptr->next == NULL)
                exit_val = 1;
        else
            current_locglobalvarptr = current_locglobalvarptr->next;
        }
    }
}

/* Go to the next record in the main link list */
exit_val = 0;
if (current_dblmpt->next == NULL)
    exit_val = 1;
else
    current_dblmpt = current_dblmpt->next;
}

if (found_locglobalvar != 1)
clear_list(globals->w2ids.list2);

/* filter the modules to be loaded to prevent redundancy existing in the database from being displayed on widget */
/* ----------------------------- */
char *filter(module)
char *module;
{

    FILE *fp;
    char *line;
    int count;

    /* Open the file containing a single sample of the module to be loaded */
    if ((fp = fopen(FILTER_FILE,"r")) == NULL) {
        perror("error opening data file");
    }
```

line = (char *)malloc(LINE_LEN);
if (line == (char *)NULL) {
    perror("not enough space on device; could not allocate memory");
}

/* Process the first line of the file */
strcpy(line,"bidon");
count = 0;
while (strcmp(line,NULL) != 0) {
    line = fgets(line,LINE_LEN,fp);
    if (strlen(line) >= 1)
        line[strlen(line)-1] = '0';
    count++;

    if ((strcmp(line,NULL) == 0) && (count == 1)) { /* The first line of the file was empty */
        /* The file was empty */
        /* Store the module on the first line of the file and return the module */
        fclose(fp);
        if ((fp = fopen(FILTER_FILE,"w")) == NULL)
            perror("error opening data file");
        fprintf(fp,"%s
",module);
        fclose(fp);
        return(module);
    }
    else if ((strcmp(line,NULL) == 0) && (count != 1)) { /* The file did not contain the module */
        /* Store the module on the last line of the file and return the module */
        fclose(fp);
        if ((fp = fopen(FILTER_FILE,"a")) == NULL)
            perror("error opening data file");
        fprintf(fp,"%s
",module);
        fclose(fp);
        return(module);
    }
    else {
        if (strcmp(module,line) == 0) { /* The module has already been loaded */
            fclose(fp);
            return(" ");
        }
    }
}
/* End of the while loop */
}

Appendix C

This appendix presents the programs used to attach a note to a Teamwork object of any kind. A description of the purpose of each function, the inputs and the outputs is given at the beginning of each program.

Files presented:

- attach_note3.batch0
- attach_note3.batch1
- makefile
- call_attach_note3.c
- attach_note3.c
# Thierry Le Gal       February 14, 1992
# attach_note3.batch0
#

# Shell script to
# - set /u/legal/twk/com as the current directory
# - opens an Xwindow in this current directory
# Note: The user can then key the name of the executable file that he wants to run in this directory

/usr/bin/X11/aixterm -geometry 108x10+0+0
   -T "Attach_note3 application"
   -fg Wheat
   -bg MidnightBlue
   -e /u/legal/twk/access/batch/attach_note3.batc1 &

# attach_note3.batch1 is a Shell script calling /u/legal/twk/com/attach_note3 and processing the exit value returned
# Thierry Le Gal February 14, 1992
# attach_note3.batch1
#
# Shell script to call the executable attach_note3 and process its exit value contained in the shell variable $?
# Is invoked by /u/legal/twk/access/batch/attach_note3.batch0

    echo "You are going to attach a note to an object in Teamwork"
    echo "
    echo "Syntax: attach_note3 -n note_file -m model_name [-o object_name] [-t type] [-c config file] [-t note title]"
    echo "  -n You must provide an object name unless attaching to a model."
    echo "  -t The types are: [-model | -dde | -dd | -erd | -ps | -matrix | -sc | -std | -asg]"
    echo ""
    exit_value=2

# Open an X window with the list of the notes available
/usr/bin/X11/aixterm -geometry 50x25+0+250
    -T "List of the notes available"
    -Ig Wheat
    -bg MidnightBlue
    -e /usr/bin/view /u/legal/twk/access/note/note_list &

# Open an X window with the list of the models available
/usr/bin/X11/aixterm -geometry 50x25+700+250
    -T "List of the models available"
    -Ig Wheat
    -bg MidnightBlue
    -e /usr/bin/view /u/legal/twk/access/note/model_list &

    while test $exit_value != 3
    do
        case $exit_value in
            0) # exit status = 0 (exit success from T/ACCESS program);
                echo "";
                echo "The Teamwork ACCESS program was runned successfully";
                echo "This X-window will disappear in 5 seconds ...";
                exit_value=3;;
            1) # exit status = 1 (exit failure from T/ACCESS program);
                echo "";
                echo "The exit value from T/ACCESS program was $exit_value"
                echo "Check what was the problem";
                echo "This X-window will disappear in 5 seconds ...";
                exit_value=3;;
            2) # exit status = 2 (exit from T/ACCESS program due to
                # a syntax error);
                echo "";
                echo "";
                echo "Enter the command with its right arguments!"
                echo "or hit <Enter> to exit the application";
                read command_line;
                if ( $command_line ) then
                    exit_value=3
                    echo "";
                    echo "";
                    echo "";
                    echo "";
                    echo "";
                    echo "";
                    echo "";
                    echo "";
                    echo "";
                    echo "";
                    echo "Exit from the application, This X-window will disappear in 5 seconds ...";
                    exit_value=3
                fi
        esac
    done
else
   /u/legal/twk/com/$command_line
   exit_value=$?
   fi;;
esac
done
sleep 5
# Makefile - to compile a Teamwork/Access program
# I added some stuff to be able to compile a program with some X windows commands
#
CC = /bin/cc
DEBUGFLAGS = -g
INCLUDES = -I$(DBA_INCLUDE)
SYSLIBS = -lm

#
DBA_INCLUDE = /cadre/lsa/include
DBA_H = $(DBA_INCLUDE)/twk_dbah
DBA_LIB = /cadre/lsa/lib
BIN_DIR = /u/legal/twk/com

ATTACH_NOTE3 = \
  attach_note3.o\n  $(DBA_LIB)/twk_lib.a

#
EXECUTABLES = \
  $(BIN_DIR)/attach_note3
#
# the works
all: $(EXECUTABLES)

attach_note3: $(BIN_DIR)/attach_note3
#
# program dependencies
$(BIN_DIR)/attach_note3: $(ATTACH_NOTE3) $(DBA_H)
$(CC) -o $(BIN_DIR)/attach_note3 $(ATTACH_NOTE3) $(SYSLIBS)
#
# The following line means that attach_note3.o needs to be remake
# if the file attach_note3.c has been changed. Or, in other words
# that attach_note3.o depends on attach_note3.c.
# The line bellow it begins with a tab character and tells the
# system how to remake attach_note3.o.
attach_note3.o: attach_note3.c
$(CC) -c $(DEBUGFLAGS) $(INCLUDES) attach_note3.c
main()
{
  int exit=0;
  int length=0;
  int number;
  char *keyboard_input;

  /* Print the syntax of the command on the screen */
  printf("Syntax: attach_note3 -n note_file -m model_name [-o object_name]
       [-type] [-c config_file] [-t note_title]\n\n     -- You must provide an object name unless attaching to a model\n     -- The types are : [ -model | -dde | -dfd | -erd | -ps | -matrix | -sc | -std | -asm ]\n\n     \n\n     -- Loop till the user wants to exit */
  printf("Enter the command with its arguments or Type 'exit' to exit\n\n     -- Loop till exit is typed */
  while(length != 4) /* four letters were keyed */
  {
    scanf("%s",&keyboard_input);
    length = strlen(&keyboard_input);

    system("/u/legal/twk/com/keyboard_input");
  }

  scanf("%d",&number);
}
#include <stdio.h> /* standard input-output header */
#include "twk_dba.h" /* Teamwork/Access header */

#define NOTE_FILE 0
#define MODEL_NAME 1
#define OBJECT_NAME 2
#define CONFIG_FILE 3
#define MAX_ARGS 4
#define DEFAULT_NOTE_TITLE "Untitled"
#define EXIT_SYNTAX 2

int syntax_error();
int parse();
int initialize();
twk_process_index_t *read_process_index();
twk_dd_index_entry_t *read_dd_index();
twk bool find_type();
void inform_user();
int validate_body_file();
int main();

int syntax_error(message, cur_arg)
char *message;
char *cur_arg;
{
    if (cur_arg != NULL)
    {
        printf("SYNTAX ERROR : %s while looking at %s\n", message, cur_arg);
        printf("\n");
        printf("\n");
        printf("Syntax: attach_note3 -n note_file -m model_name [-o object_name] [-c config_file] [-t note_title]\n");
        printf("-- You must provide an object name unless attaching to a model.\n");
        printf("-- The types are : [-model | -dde | -fd | -erd | -ps | -matrix | -sc | -std | -asg ]\n");
        exit(EXIT_SYNTAX); /* EXIT_SYNTAX is set to 2 */
    }
}

/*...........................................................................*/
```c
/* Module name: parse */
/* Description: Parse the command line with its arguments */
/* */
/* Input: The number of command line argument (argc) */
/* The command line argument (argv[]) */
/* Output: The array of pointers pointing to each argument values (args[]) */
/* The type of object to which the note will be attach (*type) */
/* found_type (*type found or not in the list) */
/* The title of the note (**title) */

int parse(argc, argv, args, type, title, found_type);
int argv_count; /* number of command line arguments */
char *argv[]; /* command line arguments */
char *args[MAX_ARGS]; /* parsed arguments */
twk_object_type_t *type; /* type option */
char **title; /* title option */
twk_bool *found_type; /* found type option boolean */
{
    int argc_count; /* count command line arguments */
    char *cur_arg= NULL; /* current command line argument */
    int i; /* counter */
    static char *OBJ_TYPE_KEY[] =
    {
        "model",
        "dote",
        "dfd",
        "erd",
        "ps",
        "matrix",
        "sc",
        "std",
        "asg",
        "sem",
        "pat",
        "dt",
        "ms",
        "m-spec",
        "p-spec",
        "blob",
        "MODEL",
        "DDE",
        "DFD",
        "ERD",
        "PS",
        "MATRIX",
        "SC",
        "STD",
        "ASG",
        "SEM",
        "PAT",
        "DT",
        "MS",
        "M-SPEC",
        "P-SPEC",
        "BLOB"
    };

    static twk_object_type_t OBJ_TYPE_TYPE[] =
    {twk_model_type,
```
static int NUM_OBJ_TYPES = 32;

/* Initialize the title to the null character */
*title = TWK_NULL; /* In twk_dba.h TWK_NULL is set to 0, this is the value of a null C string */

/* Initialize the value of each parsed argument to the null character */
for (i = 0; i < MAX_ARGS; i++)
    args[i] = TWK_NULL;

/* Initialize *found_type to false (0 in twk_dba.h) */
*found_type = TWK_FALSE;

/* Loop through each argument for the function attach_note3 */
for (argv_count = 1; argv_count < argc; argv_count++)
    {
        /* Set the value of the current command line argument */
        cur_arg = argv[argv_count];

        /* -t : get the title of the note to be created */
        if (strcmp(cur_arg, "-t") == 0) ! (strcmp(cur_arg, "-T") == 0))
            {
                if (*title == TWK_NULL) /* The -t option has already been parsed */
                    syntax_error("more than one title specified", cur_arg);

                /* Assign the title value to *title and increment the counter */
                *title = argv[++argv_count];
            }

        /* -n : get the file name of the note (file path included) */
else if (((strcmp(cur_arg, "-n") == 0)) || (strcmp(cur_arg, "-N") == 0))
{
  if (args[NOTE_FILE] != TWK_NULL) /* The -n option has already been parsed */
    syntax_error("more than one note file specified", cur_arg);

  /* Assign the note file name to args[0] and increment the counter */
  args[NOTE_FILE] = argv[++argv_count];
}

/* -m : get the model name */
else if (((strcmp(cur_arg, "-m") == 0)) || (strcmp(cur_arg, "-M") == 0))
{
  if (args[MODEL_NAME] != TWK_NULL) /* The -m option has already been parsed */
    syntax_error("more than one model name specified", cur_arg);

  /* Assign the model name to args[1] and increment the counter */
  args[MODEL_NAME] = argv[++argv_count];
}

/* -o : get the name of the object where to attach the note */
else if (((strcmp(cur_arg, "-o") == 0)) || (strcmp(cur_arg, "-O") == 0))
{
  if (args[OBJECT_NAME] != TWK_NULL) /* The -o option has already been parsed */
    syntax_error("more than one object name specified", cur_arg);

  /* Assign the object to args[2] and increment the counter */
  args[OBJECT_NAME] = argv[++argv_count];
}

/* -c : get the config file name */
else if (((strcmp(cur_arg, "-c") == 0)) || (strcmp(cur_arg, "-C") == 0))
{
  if (args[CONFIG_FILE] != TWK_NULL) /* The -c option has already been parsed */
    syntax_error("more than one config file specified", cur_arg);

  /* Assign the config file name to args[3] and increment the counter */
  args[CONFIG_FILE] = argv[++argv_count];
}

/* -type : get the object type */
else
{
  if (*found_type == TWK_TRUE) /* Has already parsed the -type */
    syntax_error("more than one type specified", cur_arg);

  /* Look through the list of OBJ_TYPE_KEY */
  for (i = 0; i < NUM_OBJTYPES; i++)
  {
    if (strcmp(cur_arg + 1, OBJ_TYPE_KEY[i]) == 0)
      /* The +1 is to pass the - sign before the type */
      {
        type = OBJ_TYPE_TYPE[i];
        *found_type = TWK_TRUE; /* Set to 1 in twk_dba.h */
        break; /* Allow the exit of the for loop when the object type is found */
      }
  }
  if (i >= NUM_OBJTYPES) /* The object type was not in the list */
    syntax_error("unrecognized option", cur_arg);

  /* End of the else statement */
}
/* End of the for loop through each argument */

/* Fill in defaults and check the final syntax */

if (args[NOTE_FILE] == TWK_NULL) /* Argument not assigned by the parser */
  syntax_error("Never found note file name", "the command line arguments");
if (args[MODEL_NAME] == TWK_NULL) /* Argument not assigned by the parser */
    syntax_error("never found model name", "the command line arguments");

if (args[OBJECT_NAME] == TWK_NULL) /* Argument not assigned by the parser */
{
    if (*found_type & & *type != tk_model_type)
    /* If the type was found in the list and was model, when the user do not
    indicate any object name, we should not exit because the function call
    tkw_attach_note in main() requires the object_name parameter to be set
    to null in this case */
        syntax_error("never found object name", "the command line arguments");
}

if (args[CONFIG_FILE] == TWK_NULL) /* Argument not assigned by the parser */
    args[CONFIG_FILE] = TWK_DEFAULT_CONFIG;

if (*title == TWK_NULL) /* Argument not assigned by the parser */
    *title = DEFAULT_NOTE_TITLE;

} /* End parse */

FLICTSPEC initialize(cf)

char *cf,
{
    twk_status_t status; /* Return status for twk commands */

    status = twk_connect(cf);
    if (status != TWK_STAT_OK)/* twk_stat_ok is set to 0 in twk_dba.h */
    {
        printf("ERROR : trying to connect to the dc server:\n");
        printf("the return value from twk_connect was %d\n", status);
        exit(EXIT_FAILURE); /* EXIT_FAILURE is set to 1 in twk_dba.h */
    }

} /* End initialize */

FLICTSPEC read_process_index

char *model_name,
{
    twk_status_t status; /* Return status for twk commands */

    /* Module name: read_process_index */
    /* Description: Get a pointer to the process index for the model_name from the Teamwork database */
    /* Input: Name of the model (model_name) */
    /* Output: Pointer to the process index (in T. database) for the model_name */

    tkw_process_index_t *read_process_index(model_name)
    char *model_name;
    {
/* twk_object_ptr_t is a union of pointers to all the different structures 
that can be retrieved to by twk_get_object */
twk_object_ptr_t object_from_dc; /* object retrieved from the Data Controller */

/* Get the pointer to the process index for the model from the database */
status = twk_get_object(twk_process_index_type, TWK_NULL, 
                      TWK_NULL, model_name, TWK_NULL, TWK_LATEST_GENERATION, 
                      &object_from_dc);

if (status != twk_stat_ok) /* twk_stat_ok is set to 0 in twk_dba.h */
{
    printf("ERROR : trying to get process index from model, \%s\n", model_name);
    printf("return value from twk_get_object was \%d\n", status);
    exit(EXIT_FAILURE); /* EXIT_FAILURE is set to 1 in twk_dba.h */
}

/* Returns the pointer to the process index */
return (object_from_dc.process_index_ptr);
} /* End of read_process_index */

="/********************************************************************************
/* Module name: read_dd_index */
*/

"********************************************************************************
/* Description: Get a pointer to the data dictionary index for the model_name from the T. database */
/* Input: Name of the model (model_name) */
/* Output: Pointer to the data dictionary index (in T. database) for the model_name */
/* End of read_dd_index */

twk_dd_index_entry_t *read_dd_index(model_name)
char *model_name,
{
    twk_status_t status; /* Return status for twk commands */
twk_object_ptr_t object_from_dc; /* Object retrieved from the Data Controller */

/* Get the pointer to the data dictionary index for the model from T. database */
status = twk_get_object(twk_dd_index_type, TWK_NULL, 
                      TWK_NULL, model_name, TWK_NULL, TWK_LATEST_GENERATION, 
                      &object_from_dc);

if (status != twk_stat_ok) /* twk_stat_ok is set to 0 in twk_dba.h */
{
    printf("ERROR : trying to get data dictionary index from model, \%s\n", model_name);
    printf("return value from twk_get_object was \%d\n", status);
    exit(EXIT_FAILURE); /* EXIT_FAILURE is set to 1 in twk_dba.h */
}

/* Return the pointer to the data dictionary index */
return (object_from_dc.dd_index_ptr);
} /* End of read_dd_index */

="/********************************************************************************
/* Module name: find_type */
/* Description: Look for an object in the process index list and the data dictionary index list having the same */
/* object name. Returns a boolean to indicate if it was found. Also read the type of the object */
/* matching with the object name */
/* End of find_type */

203
/*
    /* Input: Model name
        /* Object name
        /* Output: Boolean value to indicate if the type of the object was found
        /* Type of the object matching with the object name
*/

twk_bool find_type(model_name, object_name, type)
char *model_name;
char *object_name;
twk_object_type_t *type; /* Pointer to the structure twk_object_type_t */
{
    twk_bool found; /* found object boolean */
    twk_process_index_t *pi; /* pointer to the process index */
    twk_pi_entry_t *pi_entry_ptr; /* process index entry */
    twk_dd_index_entry_t *dd_first; /* first dd entry */
    twk_dd_index_entry_t *dd_entry_ptr; /* dd entry */
    twk_object_ptr_t *twk_object_ptr; /* needed for twk_free_object() */

    /* Assign the value TWK_FALSE (defined as 0 in T. database) to found */
    found = TWK_FALSE;

    /* If no object name was passed then attach the note to the model */
    if (object_name == TWK_NULL)
    {
        found = TWK_TRUE; /* (defined as 1 in T. database) */
        *type = twk_model_type;
        return(found);
    }

    /* Get the pointer to the process index for the model */
    pi = read_process_index(model_name);

    /* Check for the name to be in the process index list of the model */
    TWK_FOR_EACH(pi->first, pi_entry_ptr)
    /* This macro requires the pointer to the head of the list to be traversed and
     the pointer to the type of structure being traversed */
    if (strcmp(pi_entry_ptr->process_number, object_name) == 0)
        /* A process number from the process index ring matched with the object name */
    {
        found = TWK_TRUE;
        switch (pi_entry_ptr->process_type)
        {
            case (twk_process_type_dfld) : *type = twk_dfld_type; break;
            case (twk_process_type_erld) : *type = twk_erld_type; break;
            case (twk_process_type_mspec) : /* type = twk_mspec_type */ break;
            case (twk_process_type_cspec) : /* type = twk_cspec_type */ break;
            case (twk_process_type_pspec) : *type = twk_pspec_type; break;
            case (twk_process_type_std) : *type = twk_std_type; break;
            case (twk_process_type_sc) : *type = twk_sc_type; break;
            case (twk_process_type_dct) : /* type = twk_dct_type */ break;
            case (twk_process_type_pnt) : /* type = twk_pnt_type */ break;
            case (twk_process_type_ttt) : /* type = twk_ttt_type */ break;
            case (twk_process_type_ssm) : /* type = twk_ssm_type */ break;
            case (twk_process_type_asm) : /* type = twk_asm_type */ break;
            case (twk_process_type_user_obj) : *type = twk_user_obj_type; break;
            default:
                found = TWK_FALSE; break;
        } /* End of the switch */
    break; /* Exit the TWK_FOR_EACH loop */
} /* End if for checking the process index */

/* Assign the pointer to the object to be freed by twk_free_object() */
twk_object_ptr.process_index_ptr = pi;
twk_free_object(twk_process_index_type, twk_object_ptr);

if (found == TWK_TRUE) /* The object name was found in the process index ring */
    return(found);

/* Check for the name to be in the data dictionary index list of the model */
dd_first = read_dd_index(model_name); /* Get the pointer to the data dictionary */
TWK_FOR_EACH(dd_first, dd_entry_ptr)
    if (strcmp(dd_entry_ptr->dd_name, object_name) == 0)
        /* A data dictionary name from the data dictionary index list for this model */
        matched the object name */
        found = TWK_TRUE;
        *type = twk_dde_type;
        break; /* Exit the TWK_FOR_EACH loop */

/* Assign the pointer to the object to be freed by twk_free_object() */
twk_object_ptr.dd_index_ptr = dd_first;
twk_free_object(twk_dd_index_type, twk_object_ptr);

return(found);
} /* End find_type */

/* Module name: inform_user */
/* Description: Inform the user about the execution of the program */
/* Input: Name of the file containing the note (path included) */
/* Name of the model concerned */
/* Name of the object to which attach the note */
/* Title of the note to be attached */
/* Type of the object where the note is going to be attached */
/* Output: None */

void inform_user(note_file, model_name, object_name, title, type)
    char *note_file; /* File containing the note to be attached */
    char *model_name; /* Name of the model */
    char *object_name; /* Name of the object */
    char *title; /* Title of the note */
    twk_object_type_t type; /* Type of object */
{
    char *otype;

    switch (type)
    {
    case (twk_model_type): otype = "MODEL"; break;
    case (twk_dde_type): otype = "DDE"; break;
    case (twk_dfd_type): otype = "DFD"; break;
    case (twk_erd_type): otype = "ERD"; break;
    case (twk_p_spec_type): otype = "PS"; break;
    case (twk_sc_type): otype = "SC"; break;
    case (twk_std_type): otype = "STD"; break;
    case (twk_user_obj_type): otype = "BLOB"; break;
    case (twk_asg_type): otype = "ASG"; break;
    case (twk_matrix_type): otype = "MATRIX"; break;
    default: otype = "UNKNOWN"; break;
int validate_body_file(char *filename)
{
    FILE *fp = fopen(filename, "r"); /* Open the file in read mode */
    if (fp == NULL) /* The file could not be opened */
    {
        print("ERROR: The note body file %s could not be opened for read.
", filename);
        print("Unable to attach note.
");
        exit(EXIT_FAILURE);
    }
    else
    {
        fclose(fp); /* The file exists, it has been opened in read mode */
    }
}

int main(int argc, argv[]) { /* Number of arguments in the argv[] list */
    char *argv[]; /* List of the arguments */
    {
        twk_status_t status; /* Return status from twk call */
        twk_note_t *note_ptr; /* Note to be attached */
        char *new_note_number; /* Number returned by the call */
        twk_object_type_t type; /* Type of object */
        char *title; /* Note title */
        char *args[MAX_ARGS]; /* Command line arguments */
        twk_bool found_type; /* Found type of object boolean */

        /* Get the arguments and the options keyed on the command line */
        parse(argc, argv, &type, &title, &found_type);
    }
}
/* Check if the file containing the note body exists and is not empty */
validate_body_file(args[NOTE_FILE]);

/* Connect to the DC Server */
initialize(args[CONFIG_FILE]);

/* If the type option parsed was not a valid type
valid types are (model, dde, dfd, erd, ps, matrix, sc, std, asg) */
if (found_type == TWK_FALSE)
  /* Try to find an object having the same name in that model
   and then return its type if the object was found */
  found_type = find_type(args[MODEL_NAME], args[OBJECT_NAME], &type);

if (found_type == TWK_FALSE) /* The object could not be found */
{
  printf("ERROR: Could not find the object, %s, in the model, %s.\n", 
         args[OBJECT_NAME], args[MODEL_NAME]);

  /* Disconnect from the DC Server */
  twk_disconnect();

  exit(EXIT_FAILURE);
}

/* Inform the user */
inform_user(args[NOTE_FILE], args[MODEL_NAME], args[OBJECT_NAME], title, type);

/* Allocate some memory for the note to be created */
if ( ((note_ptr = (twk_note_t *) malloc(sizeof(twk_note_t))) ) )
  /* malloc() returns the pointer null if the allocation failed */
  /* In case the null pointer is returned, the logical expression
      becomes if ( 0 ) <=> if ( 1 ) end we enter the if statement */
  
  printf("ERROR: Could not malloc note_ptr.\n");
  exit(EXIT_FAILURE);

/* Set the note in the data structure */
strcpy(note_ptr->note_title, title);
note_ptr->data_check_field = twk_data_type_note;
note_ptr->note_type = twk_note_type_text;
note_ptr->note_body_file_name = args[NOTE_FILE];

/* Attach the note (takes one call) */
status = twk_attach_note(note_ptr, "", type,
                        TWK_NULL, TWK_NULL,
                        args[MODEL_NAME], args[OBJECT_NAME], TWK_LATEST_GENERATION,
                        &new_note_number);

if (status != twk_stat_ok)
{
  printf("ERROR: trying to attach note.\n");
  printf("return value %d from twk_attach_note\n", status);
  exit(EXIT_FAILURE);
}

printf("The assigned note number is %s\n", new_note_number);

free(note_ptr);

twk_disconnect();
exit(EXIT_SUCCESS); /* EXIT_SUCCESS is set to 0 in twk_dba.h */
} /* End main */
Appendix D

This appendix presents the programs used to create some Teamwork User Menus to run some features provided by the Integration CASE workbench. A description of the purpose of each function, the inputs and the outputs is given at the beginning of each program.

Files presented:

- utility.menu
- mi.menu
- crev.menu
- utility1.menu
- toolkit.menu
- desktop.menu
- crev.menu
# Thierry Le Gal

# utility.menu

# Contains variables for dumping one or all the models from Teamwork database to a flat file. Also contains the # menu item "Utility" appended to the desktop menu bar.


#------------------------------------ Variables to dump the entire database -------------------------------------

(Variable (Id DUMP_TSA_PARAMS_ALL)
  (Value "%t_CONFIG_FILE ALL -all -d %FILE_NAME -complete &")
)

# Get the file name into which the database is going to be dumped
(Variable (Id GET_FILE_NAME_ALL)
  (Value "%T_ASSIGN(%FILE_NAME, %tENTER_FORM(%STRING(DUMP_TSA - Flat File), %STRING(get of the file where the database is going to be dumped), %DEFAULT_FILE_NAME))"
)
)

(Variable (Id DEFAULT_FILE_NAME) (Value "/u/legal/twk/junk1")

# Run dump_tsa
(Variable (Id RUN_DUMP_TSA_ALL)
  (Value "%DUMP_TSA READY MSG_ALL %SYS_CALL(%STRING(%DUMP_TSA_COMMAND_ALL))"
)
)

(Variable (Id DUMP_TSA_COMMAND_ALL)
  (Value "%t CADRE_DIR/admin/dump_tsa %DUMP_TSA_PARAMS_ALL"
)

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (Id DUMP_TSA_READY_MSG_ALL)
  (Value "%t ENTER_FORM(%STRING(DUMP_TSA), %STRING(About to run dump_tsa)

  Dumping the entire database from Teamwork database
  Into the flat file %FILE_NAME;
  Using the config file %T_CONFIG_FILE;
  Failure Log in /u/leg/twk/error_log

  Click OK to continue.
  Click CANCEL to abort dump_tsa., %NULL"
)
)

#------------------------------------ Variables used to dump a model -----------------------------------

# Get the inputs from the user
(Variable (Id GET_INPUT)
  (Value "%T MODEL_NAME %GET_FILE_NAME"
)
)

# Get the file name into which the model is going to be dumped
(Variable (Id GET_FILE_NAME)
  (Value "%T_ASSIGN(%FILE_NAME, %tENTER_FORM(%STRING(DUMP_TSA - Flat File),

210
%%% STRING(Name of the file where the selected model is going to be dumped),
%%% DEFAULT_FILE_NAME)) *
%
}
(Variable (ld DEFAULT_FILE_NAME) (Value "%/legal/twk/junk1")

# Make sure the user has selected a model and if he has return its name.
(Variable (ld MODEL_NAME)
  (Value "%T.INFORM_FORM(An Object must be selected),
  %STRING(Selected Object must be a Model)) %RETURN")
)

# If the selected object is not a DFD, this tells the user and returns.
(Variable (ld OBJECT NOT MODEL)
  (Value "%T.INFORM_FORM(An Object must be selected),
  %STRING(Selected Object must be a Model)) %RETURN")
)

(Variable (ld VALUE MODEL NAME)
  (Value "%T.INFORM_FORM(Value of model name),
  %MODEL NAME) %RETURN")
)

# Run dump_tsa
(Variable (ld RUN_DUMP TSA)
  (Value "%DUMP_TSA READY MSG
  %SYS_CALL(%STRING(%DUMP_TSA COMMAND))")
)

(Variable (ld DUMP TSA COMMAND)
  (Value "%T.CADRE DIR/admin/dump_tsa %DUMP_TSA PARAMS")
)

# Note that the parameters in single quotes are case sensitive
(Variable (ld DUMP_TSA PARAMS)
  (Value "-config %T.CONFIG_FILE -model %MODEL NAME -d %FILE_NAME -complete &")
)

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (ld DUMP TSA READY MSG)
  (Value "%T.ENTER FORM(Confirmation Request),
  %STRING(About to run dump_tsa)

Dumping the model "%MODEL NAME" from Teamwork database
Into the flat file "%FILE NAME".
Using the config file "%T.CONFIG_FILE".
Failure Log in /u/legal/twk/error_log

Click OK to continue.
Click CANCEL to abort dump_tsa.

%
)

#-------- Variables used to load in the database a dumpfile containing some (1 or more) dumped models ---------

# Syntax : load_tsa dumpfile filename [-config config_file]
# [-all] [-complete] [-prompt]
# loads all models from the dumped file, if the model already exist, prompts the
# user to replace or skip the model. If the model does not exist, it asks the user
# to enter a new name for it

# Get the file name of the dumped file to load into teamwork
(Variable (ld GET FILE NAME TO LOAD)
(Value "%ASSIGN("FILE_NAME_TO_LOAD";%ENTER_FORM(%STRING(LOAD TSA - Flat File),
   %STRING(Name of the dumped file containing the models to be loaded),
   %DEFAULT_FILE_NAME_TO_LOAD))")
)
)
(Variable (Id DEFAULT_FILE_NAME_TO_LOAD) (Value "/u/lega/twk/junk1") )

# Run load_tsa
(Variable (Id RUN_LOAD_TSA)
   (Value "%LOAD_TSA_READY_MSG
       %SYS_CALL(%STRING(%LOAD_TSA_COMMAND))")
)

(Variable (Id LOAD_TSA_COMMAND)
   (Value "%t.CADRE_DIR/admin/load_tsa %LOAD_TSA_PARAMS")
)

# Note that the parameters in single quotes are case sensitive
(Variable (Id LOAD_TSA_PARAMS)
   (Value "-dumpfile %FILE_NAME_TO_LOAD -config %t.CONFIG_FILE -all -complete -prompt &")
)

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (Id LOAD_TSA_READY_MSG)
   (Value "%t.ENTER_FORM(%STRING(LOAD_TSA),
       %STRING(About to run load_tsa

   Loading the models contained in the dump file %FILE_NAME_TO_LOAD
   Using the config file %t.CONFIG_FILE.
   Failure Log in /u/lega/twk/error_log

   Click OK to continue.
   Click CANCEL to abort dump_tsa.), %NULL")
)

#----------------------------------------------------------
# This menu enable you to dump one model or the entire database from Teamwork database
#----------------------------------------------------------

(Menu
   (Name "Tt_Utilities")
   (MenuItem
      (Name "Dump a model")
      (Action (SysCall "%GET_INPUT
                   %RUN_DUMP_TSA")
   )
   (MenuItem
      (Name "Dump the database")
      (Action (SysCall "%GET_FILE_NAME_ALL
                   %RUN_DUMP_TSA_ALL")
   )
   (MenuItem
      (Name "Load a dump file")
      (Action (SysCall "%GET_FILE_NAME_TO_LOAD
                   %RUN_LOAD_TSA")
   )
)
# Thierry Le Gal
# mi.men

# Top level menus for including menus in the Teamwork mi

(Include "%t.CADRE_DIR/menus/general/twk_base.men")
(Include "%t.CADRE_DIR/menus/mi/list.men")
(Include "%t.CADRE_DIR/menus/mi/mi_cslib.men")
(Include "%t/legal/twk/menus/mi/utility.menu")
(Include "%t/legal/twk/menus/mi/cprev.menu")
# Thierry Le Gal  March 04, 1992
# crev.menu
#
# Contains variables for creating some Structure Charts
# - Generate some SCs by running T/Crev on C source code (a treed is generated)
# - Generate a tree for all the levels below a particular function and generate some SCs by running T/Crev on
#   the considered call tree
# - Generate some SCs by running T/Crev on a call tree
#
#
# Syntax /cadre/tool_kit/crev [-config config_file] [-files filename_list]
# [-model model_name] [-title string] [-call_tree call_file]
# When the -files option is used, -call_tree creates a intermediate treed
#
#
# Variables to generate some SCs by running T/Crev on C source code (a treed is also created)
#
# Get the file name where to put the generated call tree
# (Variable (id GET_CTREE_NAME_CODE)
# (Value "%ASSIGN(%CTREE_NAME_CODE,%ENTER_FORMAT(%STRING(Create SC from code),
#   %STRING(File name where to put the call tree),
#   %DEFAULT_CTREE_NAME_CODE))")
#)
# (Variable (id DEFAULT_CTREE_NAME_CODE) (Value "/u/legals/wk/calitree/ctree_%SOURCE_FILE_NAME") )

# Make sure the user has selected a model and if he has return its name.
# (Variable (id GET_MODEL_NAME_CODE)
# (Value "%IF(%EQ(%SELECTED_OBJECT_TYPE,%SELECTED_OBJECT_TYPE_MODEL),
#   %SELECTED_OBJECT,%SELECTED_OBJECT_NOT_MODEL)")
#)

# If the selected object is not a DFD, this tells the user and returns.
# (Variable (id SELECTED_OBJECT_NOT_MODEL)
# (Value "%INFORM_FORMAT(%STRING(Create SC from code),
#   %STRING(Select a model where to put the structure charts)) %RETURN")
#)

# Get the name of the config file to be used
# (Variable (id GET_CF_NAME)
# (Value "%ASSIGN(%CF_NAME,%ENTER_FORMAT(%STRING(Create SC from code),
#   %STRING(Config file to be used),
#   %DEFAULT_CF_NAME))"
#)
#)
# (Variable (id DEFAULT_CF_NAME) (Value "/u/legals/wk/tsc/1t.cf_db") )

# Get the path of the source file on which to run T/Crev
# (Variable (id GET_SOURCE_FILE_PATH)
# (Value "%ASSIGN(%SOURCE_FILE_PATH,%ENTER_FORMAT(%STRING(Create SC from code),
#   %STRING(Path of the source code),
#   %DEFAULT_SOURCE_FILE_PATH))"
#)
#)
# (Variable (id DEFAULT_SOURCE_FILE_PATH) (Value "/u/legals/wk/calltree/source/" ) )

214
# Get the name of the source file on which to run T/Crev
(Variable (Id GET_SOURCE_FILE_NAME)
  (Value "%ASSIGN(%SOURCE_FILE_NAME,%t.ENTER_FORM(%STRING(Create SC from code),
  %STRING(File name of the source code),
  %DEFAULT_SOURCE_FILE_NAME))"
)
)
(Variable (Id DEFAULT_SOURCE_FILE_NAME) (Value "pr_input.c")
)

# Defines the source file
(Variable (Id SOURCE_FILE)
  (Value "%SOURCE_FILE_PATH%SOURCE_FILE_NAME"
)
)

# Get the title for the appli SC
(Variable (Id GET_TITLE_SC_CODE)
  (Value "%ASSIGN(%TITLE_CODE,%t.ENTER_FORM(%STRING(Create SC from code),
  %STRING(Title for the SC),
  %DEFAULT_TITLE_SC_CODE))"
)
)
(Variable (Id DEFAULT_TITLE_SC_CODE) (Value "T/Crev_%SOURCE_FILE_NAME")
)

# Run th T/Crev command
(Variable (Id RUN_CREV_COMMAND_CODE)
  (Value "%CREV_READY_MSG_CODE
  %SYS_CALL(%STRING(%CREV_COMMAND_CODE))"
)
)

# T/Crev command
(Variable (Id CREV_COMMAND_CODE)
  (Value "%CADRE_DIR/tool_kit/crev %CREV_PARAMS_CODE"
)
)

# Note that the parameters in single quotes are case sensitive
(Variable (Id CREV_PARAMS_CODE)
  (Value "--config %CF_NAME''--model %GET_MODEL_NAME_CODE'--call_tree %CTREE_NAME_CODE'--files \n  %SOURCE_FILE'--title %TITLE_CODE &")
)

# Define the variables
(Variable (Id CTREE_NAME) (Value "")
(Variable (Id MODEL_NAME_CODE) (Value "")
(Variable (Id CF_NAME) (Value "")
(Variable (Id SOURCE_FILE) (Value "")
(Variable (Id SOURCE_FILE_PATH) (Value "")
(Variable (Id SOURCE_FILE_NAME) (Value "")
(Variable (Id TITLE_CODE) (Value "")

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (Id CREV_READY_MSG_CODE)
  (Value "%t.ENTER_FORM(%STRING(Create SC from code),
  %STRING(About to run T/Crev

Using the source code %SOURCE_FILE_PATH%SOURCE_FILE_NAME'
Using the config file %CF_NAME'
Creating the call tree %CTREE_NAME_CODE',
Creating the SC in the model %GET_MODEL_NAME_CODE'
Failure Log in /u/legal/twk/error_log

Click OK to continue.
Click CANCEL to abort this command), %NULL)."
)
)

# Prompt for the end of the application
(Variable #(id END_MSG_CODE)
  (Value "%I.INFORM_FORM(%STRING(End!!!),
    %STRING(Go and check the SC in the model %GET_MODEL_NAME_CODE)) %RETURN")
)

# Variables to generate some SCs by running T/Cre on a ctree obtained for all the levels below a given function
#-----------------------------------------------------------------------------------

# Make sure the user has selected a model and if he has return its name.
(Variable #(id GET_MODEL_NAME_DEF)
  (Value "%I.IF(%EQ(%SELECTED_OBJECT_TYPE,%SELECTED_OBJECT_TYPE_MODEL),
    %SELECTED_OBJECT,%SELECTED_OBJECT_NOT_MODEL_DEF")
)

# If the selected object is not a DFD, this tells the user and returns.
(Variable #(id SELECTED_OBJECT_NOT_MODEL_DEF)
  (Value "%I.INFORM_FORM(%STRING(Create SC below function),
    %STRING(Select a model where to put the structure charts)) %RETURN")
)

# Get the name of the call tree to be trimmed
(Variable #(id GET_CTREE_NAME_TO_BE_TRIMMED_DEF)
  (Value "%I.ASSIGN(%CTREE_NAME_TO_BE_TRIMMED_DEF,%ENTER_FORM(%STRING(Create SC below function),
    %STRING(Name of the call tree to be trimmed),
    %DEFAULT_CTREE_NAME_TO_BE_TRIMMED_DEF))
)
)
(Variable #(id DEFAULT_CTREE_NAME_TO_BE_TRIMMED_DEF) (Value "%/u/legal/twk/calltree/ctree1.c")
)

# Get the name of the function for which we want to build a ctree definition
(Variable #(id GET_FUNCTION_NAME_DEF)
  (Value "%I.ASSIGN(%FUNCTION_NAME_DEF,%ENTER_FORM(%STRING(Create SC below function),
    %STRING(Name of the considered function),
    %DEFAULT_FUNCTION_NAME_DEF))
)
)
(Variable #(id DEFAULT_FUNCTION_NAME_DEF) (Value "%STRING(proc_input)")
)

# Get the title for the SC generated
(Variable #(id GET_TITLE_SC_DEF)
  (Value "%I.ASSIGN(%TITLE_DEF,%ENTER_FORM(%STRING(Create SC below function),
    %STRING(Title for the structure charts),
    %DEFAULT_TITLE_SC_DEF))
)
)
(Variable #(id DEFAULT_TITLE_SC_DEF) (Value "T/Crev_%FUNCTION_NAME_DEF")
)

# Get the name of the config file to be used
(Variable #(id GET_CF_NAME_DEF)
  (Value "%I.ASSIGN(%CF_NAME_DEF,%ENTER_FORM(%STRING(Create SC below function),
    %STRING(Config file to be used),
    %DEFAULT_CF_NAME_DEF))
)
)
(Variable #(id DEFAULT_CF_NAME_DEF) (Value "%/u/legal/twk/lsa/l_of_db")
)
# Run the batch to trim the ctree and run T/Crev
(Variable (id RUN_TRIM_CTREE_FCT_DEF_AND_CREATE_SC)
  (Value "%READY_MSG_TRIM_CTREE_FCT_DEF
  %SYS_CALL(%STRING(%TRIM_CTREE_FCT_DEF_AND_CREATE_SC))")
)

# Batch to trim the ctree and run T/Crev
(Variable (id TRIM_CTREE_FCT_DEF_AND_CREATE_SC)
  (Value "bin/ksh /u/legal/twk/calltree/awk/start.def %BATCH_PARAMS_DEF")
)

# Parameters passed to the batch
(Variable (id BATCH_PARAMS_DEF)
  (Value "%CTREE_NAME_TO_BE_TRIMMED_DEF" %FUNCTION_NAME_DEF %GET_MODEL_NAME_DEF
    %CF_NAME %TITLE_DEF &")
)

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (id READY_MSG_TRIM_CTREE_FCT_DEF)
  (Value "%ENTER_FORM(%STRING(Create SC below function),
    %STRING(About to trim the ctree and generate the structure charts)
    Trim the call tree %CTREE_NAME_TO_BE_TRIMMED_DEF
    Create the file /u/legal/twk/calltree/ctree_%FUNCTION_NAME_DEF%QUOTE(.def)
    Store the structure charts in the model %GET_MODEL_NAME_DEF
    Using the config file %CF_NAME_DEF")
  Failure Log in /u/legal/twk/error_log

  Click OK to continue.
  Click CANCEL to abort this command). %NULL")
)

# Prompt for the end of the application
(Variable (id END_MSG_TRIM_CTREE_FCT_DEF)
  (Value "%INFORM_FORM(%STRING(End !!!),
    %STRING(Check the SCs in the model %GET_MODEL_NAME_DEF)) %.RETURN")
)

# Define the variables
(Variable (id CTREE_NAME_TO_BE_TRIMMED_DEF) (Value "")
(Variable (id FUNCTION_NAME_DEF) (Value "")
(Variable (id TITLE_DEF) (Value "")
(Variable (id CF_NAME_DEF) (Value "")

# Variables to generate some SCs by running T/Crev on a ctree obtained for some levels around a given function
#--------------------------------------------------

# Make sure the user has selected a model and if he has return its name.
#(Variable (id GET_MODEL_NAME_LEVEL)
#  (Value "%IF(%EQ(%SELECTED_OBJECT_TYPE,%SELECTED_OBJECT_TYPE_MODEL),
#    %SELECTED_OBJECT,%SELECTED_OBJECT NOT MODEL_LEVEL")
#)

# If the selected object is not a DFD, this tells the user and returns.
(Variable (id SELECTED_OBJECT_NOT_MODEL_LEVEL)
  (Value "%INFORM_FORM(%STRING(Create SC around function),
    %STRING(Select a model where to put the structure charts)) %.RETURN")

217
) (Variable (id GET_MODEL_NAME_LEVEL) (Value "%ASSIGN(%MODEL_NAME_LEVEL,%t.ENTER_FORM(%STRING(Create SC around function), %STRING(Name of the model), %DEFAULT_MODEL_NAME_LEVEL)) ")
)

) (Variable (id DEFAULT_MODEL_NAME_LEVEL) (Value "ti_complete_appl") ) (Variable (id MODEL_NAME_LEVEL) (Value "") )

# Get the name of the call tree to be trimmed (Variable (id GET_CTREE_NAME_TO_BE_TRIMMED_LEVEL) (Value "%ASSIGN(%CTREE_NAME_TO_BE_TRIMMED_LEVEL,%t.ENTER_FORM(%STRING(Create SC around function), %STRING(Name of the call tree to be trimmed), %DEFAULT_CTREE_NAME_TO_BE_TRIMMED_LEVEL)) ")
)

) (Variable (id DEFAULT_CTREE_NAME_TO_BE_TRIMMED_LEVEL) (Value "/user/legale/twk/calltree/calltree1.c") )

# Get the name of the function for which we want to build a ctree definition (Variable (id GET_FUNCTION_NAME_LEVEL) (Value "%ASSIGN(%FUNCTION_NAME_LEVEL,%t.ENTER_FORM(%STRING(Create SC around function), %STRING(Name of the considered function), %DEFAULT_FUNCTION_NAME_LEVEL)) ")
)

) (Variable (id DEFAULT_FUNCTION_NAME_LEVEL) (Value "%STRING(get_input)") )

# Get the number of levels above the function to represent (Variable (id GET_NUMBER_LEVEL_ABOVE) (Value "%ASSIGN(%NUMBER_LEVEL_ABOVE,%t.ENTER_FORM(%STRING(Create SC around function), %STRING(Number of levels above the function to be represented), %DEFAULT_NUMBER_LEVEL_ABOVE)) ")
)

) (Variable (id DEFAULT_NUMBER_LEVEL_ABOVE) (Value "0") )

# Get the number of levels below the function to represent (Variable (id GET_NUMBER_LEVEL_Below) (Value "%ASSIGN(%NUMBER_LEVEL_Below,%t.ENTER_FORM(%STRING(Create SC around function), %STRING(Number of levels below the function to be represented), %DEFAULT_NUMBER_LEVEL_Below)) ")
)

) (Variable (id DEFAULT_NUMBER_LEVEL_Below) (Value "5") )

# Get the title for the SC generated (Variable (id GET_TITLE_SC_LEVEL) (Value "%ASSIGN(%TITLE_LEVEL,%t.ENTER_FORM(%STRING(Create SC around function), %STRING(Title for the structure chart), %DEFAULT_TITLE_SC_LEVEL)) ")
)

) (Variable (id DEFAULT_TITLE_SC_LEVEL) (Value "T/Crev,%FUNCTION_NAME_LEVEL") )

# Get the name of the config file to be used (Variable (id GET_CF_NAME_LEVEL) (Value "%ASSIGN(%CF_NAMELEVEL,%t.ENTER_FORM(%STRING(Create SC around function), %STRING(Config file to be used), %DEFAULT_CF_NAME_LEVEL)) ")
)

)
# Run the batch to trim the ctree and run T/Crew
(Variable (id RUN_TRIM_CTREE_FCT_LEVEL_AND_CREATE_SC)
 (Value "%%READY_MSG_TRIM_CTREE_FCT_LEVEL
  %SYS_CALL(%STRING(%TRIM_CTREE_FCT_LEVEL_AND_CREATE_SC))"
 )
)

# Batch to trim the ctree and run T/Crew
(Variable (id TRIM_CTREE_FCT_LEVEL_AND_CREATE_SC)
 (Value "$/bin/ksh $/legal/twk/calltree/awk/start.level.copy %BATCH_PARAMS_LEVEL"
 )
)

# Parameters passed to the batch
(Variable (id BATCH_PARAMS_LEVEL)
 (Value "%CTREE_NAME_TO_BE_TRIMMED_LEVEL' %FUNCTION_NAME_LEVEL' %NUMBER_LEVEL_ABOVE
  %NUMBER_LEVEL_BELOW' %MODEL_NAME_LEVEL' %CF_NAME_LEVEL' %TITLE_LEVEL' &")
 )

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (id READY_MSG_TRIM_CTREE_FCT_LEVEL)
 (Value "%ENTER_FORM%STRING(Create SC around function),
  %STRING(About to trim the ctree and generate the structure charts

Trim the call tree %CTREE_NAME_TO_BE_TRIMMED_LEVEL'
Around the function %FUNCTION_NAME_LEVEL' %NUMBER_LEVEL_ABOVE' levels above
%NUMBER_LEVEL_BELOW' levels below
Create the file $/legal/twk/calltree/ctree/cf%FUNCTION_NAME_LEVEL%QUOTE(level)
Store the structure charts in the model %MODEL_NAME_LEVEL'
Using the config file %CF_NAME_LEVEL'

Failure Log in $/legal/twk/error_log

Click OK to continue.
Click CANCEL to abort this command), %NULL"
 )

# Define the variables
(Variable (id CTREE_NAME_TO_BE_TRIMMED_LEVEL) (Value "")
(Variable (id FUNCTION_NAME_LEVEL) (Value "")
(Variable (id NUMBER_LEVEL_ABOVE) (Value "")
(Variable (id NUMBER_LEVEL_BELOW) (Value "")
(Variable (id TITLE_LEVEL) (Value "")
(Variable (id CF_NAME_LEVEL) (Value "")

# Variables used to generate some SCs by running T/Crew on a call tree
#--------------------------
# Syntax /cadre/toolkit/crev [-config config_file] [-model model_name] [-title string] [-call_tree_call_file]
#--------------------------

# Get the path of the source file on which to run T/Crew
(Variable (id GET_CTREE_PATH_CTREE)
 (Value "$ASSIGN(%CTREE_PATH_CTREE,$DELETE_FORM(%STRING(Create SC from call tree),
  %STRING(Path of the directory containing the call tree),
  %DEFAULT_CTREE_PATH_CTREE))"
 )

219
(Variable (id DEFAULT_CTREE_PATH_CTREE) (Value "//legal/twk/calltree/ctree") )

# Get the file name where is stored the call tree
(Variable (id GET_CTREE_NAME_CTREE) (Value "%ASSIGN(%CTREE_NAME_CTREE,%t.ENTER_FORM(%STRING(Create SC from call tree), %STRING(File name of the call tree to be read), %DEFAULT_CTREE_NAME_CTREE))"
  )
)
(Variable (id DEFAULT_CTREE_NAME_CTREE) (Value "") )

# Make sure the user has selected a model and if he has return its name.
(Variable (id GET_MODEL_NAME_CTREE) (Value "%IF(%EQ(%SELECTED_OBJECT_TYPE,%SELECTED_OBJECT_TYPE_MODEL), %SELECTED_OBJECT,%SELECTED_OBJECT_NOT_MODEL_CTREE)"
  )
)

# If the selected object is not a DFD, this tells the user and returns.
(Variable (id SELECTED_OBJECT_NOT_MODEL_CTREE) (Value "%INFORM_FORM(%STRING(Create SC from call tree), %STRING(Select a model where to put the structure charts)) %RETURN"
  )
)

# Get the title for the SC generated
(Variable (id GET_TITLE_SC_CTREE) (Value "%ASSIGN(%TITLE_CTREE,%t.ENTER_FORM(%STRING(Create SC from call tree), %STRING(Title for the SC), %DEFAULT_TITLE_SC_CTREE))"
  )
)
(Variable (id DEFAULT_TITLE_SC_CTREE) (Value "T/Crev_%CTREE_NAME_CTREE") )

# Get the name of the config file to be used
(Variable (id GET_CF_NAME_CTREE) (Value "%ASSIGN(%CF_NAME_CTREE,%t.ENTER_FORM(%STRING(Create SC from call tree), %STRING(Config file to be used), %DEFAULT_CF_NAME_CTREE))"
  )
)
(Variable (id DEFAULT_CF_NAME_CTREE) (Value "//legal/twk/tsa/tl_cf_db") )

# Run the T/Crev command
(Variable (id RUN_CREV_COMMAND_CTREE) (Value "%CREV_READY_MSG_CTREE %SYS_CALL(%STRING(%CREV_COMMAND_CTREE))"
  )
)

# T/Crev command
(Variable (id CREV_COMMAND_CTREE) (Value "%CADRE_DIR/tool_kit/crev %CREV_PARAMS_CTREE"
  )
)

# Note that the parameters in single quotes are case sensitive
(Variable (id CREV_PARAMS_CTREE) (Value "-config %CF_NAME_CTREE -model %GET_MODEL_NAME_CTREE -call_tree %CTREE_PATH_CTREE %CTREE_NAME_CTREE -title %TITLE_CTREE &")
  )

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (id CREV_READY_MSG_CTREE)
 (Value "%st.ENTER_FORM(STRING(Create SC from call tree),
 STRING(About to run T/Crev

Using the ctree "%CTREE_PATH_CTREE" + %CTREE_NAME_CTREE"
Using the config file `%CF_NAME_CTREE`
Creating the SC in the model `%GET_MODEL_NAME_CTREE`
Failure Log in /u/legal/wk/error_log

Click OK to continue.
Click CANCEL to abort this command), %NULL)
)

# Prompt for the end of the application
(Variable (id END_MSG_CTREE)
 (Value "%st.INFORM_FORM(STRING(End !!!),
 STRING(Go and check the SC in the model `%GET_MODEL_NAME_CTREE`)) %RETURN)
)

# Define the variables
(Variable (id CTREE_NAME_CTREE) (Value ""))
(Variable (id CTREE_PATH_CTREE) (Value ""))
(Variable (id MODEL_NAME_CTREE) (Value ""))
(Variable (id TITLE_CTREE) (Value ""))
(Variable (id CF_NAME_CTREE) (Value ""))

# For debugging purpose
#(Variable (id Verbose) (Value "%TRUE"))
#(Variable (id Verbose) (Value "%Exit_Info"))

#-----------------------------------------------------------
# Menu definition
# - Create.SC_from_code: Generate some SCs by running T/Crev on C source code (a ctree is generated)
# - Create.SC_below_fct: Generate a ctree for all the levels below a particular function and generate some SCs
# by running T/Crev on the considered ctree
# - Create.SC_from_ctree: Generate some SCs by running T/Crev on a call tree
#-----------------------------------------------------------

(Menu
 (Name "%T_Create.SC")
 (MenuItem
  (Name "Create.SC_from_code")
  (Action (SysCall "%GET_MODEL_NAME_CODE
  %GET_SOURCE_FILE_PATH
  %GET_SOURCE_FILE_NAME
  %GET_CTREE_NAME_CODE
  %GET_TITLE.SC_CODE
  %GET_CF_NAME
  %RUN_CREV_COMMAND_CODE")))
)

(MenuItem
 (Name "Create.SC_below_fct")
 (Action (Syscall"%GET_MODEL_NAME_DEF
 %GET_CTREE_NAME_TO_BE_TRIMMED_DEF
 %GET_FUNCTION_NAME_DEF
 %GET_TITLE.SC_DEF
 %GET_CF_NAME_DEF
 %RUN_TRIM_CTREE_FCT_DEF_AND_CREATE.SC")))
)
(Menultem
  (Name "Create_SC_around_fct")
  (Action (Syscall "%GET_MODEL_NAME_LEVEL
             %GET_CTREE_NAME_TO_BE_TRIMMED_LEVEL
             %GET_FUNCTION_NAME_LEVEL
             %GET_NUMBER_LEVEL_ABOVE
             %GET_NUMBER_LEVEL_BELOW
             %GET_TITLE_SC_LEVEL
             %GET_CF_NAME_LEVEL
             %RUN_TRIM CTREE_FCT_LEVEL_AND_CREATE_SC"])})

(Menultem
  (Name "Create_SC_from_ctree")
  (Action (Syscall "%GET_MODEL_NAME_CTREE
              %GET_CTREE_PATH_CTREE
              %GET_CTREE_NAME_CTREE
              %GET_TITLE_SC_CTREE
              %GET_CF_NAME_CTREE
              %RUN_CREV_COMMAND_CTREE"])})}
# Contains variables for dumping one or all the models from Teamwork database to a flat file
# Also contains the menu item "Utility" appended to the desktop menu bar.

# Syntax: DUMP_TSA -dumpfile filename [-config config_file]
#        [-all | [-model name...]] [-complete]
#        [-help [help_dir]] [-syntax [help_dir] -version] [-options]
#        [-arg_input_filename] [-noheader] [-quiet] [-ask]
#        [-unix] [-vms] [-os2]

# This way of programming presents the advantage of being able to define
# a default value for the file name and for the model name

# Get the inputs from the user
(Variable (ld GET_INPUT)
 (Value "%GET_MODEL_NAME_U
 %GET_FILE_NAME_U")
 )

# Get the file name into which the model is going to be dumped
(Variable (ld GET_FILE_NAME_U)
 (Value "%ASSIGN(%FILE_NAME,%t.ENTER_FORM(%STRING(DUMP_TSA - Flat File),
 % STRING(Name of the file where the model is going to be dumped),
 %DEFAULT_FILE_NAME))"
 )
 )
(Variable (ld DEFAULT_FILE_NAME) (Value "/u/legal/twk/junk1") )

# Get the model name of the model to be dumped
(Variable (ld GET_MODEL_NAME_U)
 (Value "%ASSIGN(%MODEL_NAME,%t.ENTER_FORM(%STRING(DUMP_TSA - Model),
 % STRING(Model to be dumped),
 %DEFAULT_MODEL_NAME))"
 )
 )
(Variable (ld DEFAULT_MODEL_NAME) (Value "tl_Cruise_Control") )

# Run dump_tsa
(Variable (ld RUN_DUMP_TSA)
 (Value "%DUMP_TSA_READY_MSG
 %SYS_CALL(%STRING(%DUMP_TSA_COMMAND))")
 )
(Variable (ld DUMP_TSA_COMMAND)
 (Value "%t.CADRE_DIR/admin/dump_tsa %DUMP_TSA_PARAMS")
 )

# Note that the parameters in single quotes are case sensitive
(Variable (ld DUMP_TSA_PARAMS)
 (Value "-config %t.CONFIG_FILE -model %MODEL_NAME -d %FILE_NAME -complete &")
 )

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (ld DUMP_TSA_READY_MSG)
 (Value "%t.ENTER_FORM(%STRING(DUMP_TSA),
 %STRING(About to run dump_tsa"

Dumping the model '%MODEL_NAME' from Teamwork database
Into the flat file `%FILE_NAME`.
Using the config file `%t.CONFIG_FILE`.
Failure Log in `/u/legal/twk/error_log`

Click OK to continue.
Click CANCEL to abort dump_tsa.), %NULL)*

#---------------------------------------------------------- Variables to dump the entire database -----------------------------------------------
(Variable (Id DUMP_TSA_PARAMS_ALL)
 (Value `*config %t.CONFIG_FILE -all -d `%FILE_NAME` -complete &`)
)

# Get the file name into which the database is going to be dumped
(Variable (Id GET_FILE_NAME_U_ALL)
 (Value `%ASSIGN('%FILE_NAME','%t.ENTER_FORM(%STRING(DUMP_TSA - Flat File),
   %STRING(Name of the file where the database is going to be dumped),
   %DEFAULT_FILE_NAME))`)
)
(Variable (Id DEFAULT_FILE_NAME) (Value `*/u/legal/twk/junk1`) )

# Run dump_tsa
(Variable (Id RUN_DUMP_TSA_ALL)
 (Value `%DUMP_TSA_READY_MSG_ALL
   %SYS_CALL(%STRING(%DUMP_TSA_COMMAND_ALL))`)
)
(Variable (Id DUMP_TSA_COMMAND_ALL)
 (Value `%t.CADRE_DIR/admin/dump_tsa %DUMP_TSA_PARAMS_ALL`)
)

# Before we run this tell the user what he is about to do and ask if it is OK.
(Variable (Id DUMP_TSA_READY_MSG_ALL)
 (Value `%t.ENTER_FORM(%STRING(DUMP_TSA),
   %STRING(About to run dump_tsa

Dumping the entire database from Teamwork database
Into the flat file `%FILE_NAME`;
Using the config file `%t.CONFIG_FILE`;
Failure Log in `/u/legal/twk/error_log`

Click OK to continue.
Click CANCEL to abort dump_tsa.), %NULL)*

#----------------------------------------------------------
# This menu enable you to dump one model or the entire database from Teamwork database
#----------------------------------------------------------

(Menu
 (Name "Tl_Utilities")
 (Menuitem
   (Name "Dump a model")
   (Action (SysCall `%GET_INPUT
     %RUN_DUMP_TSA`))
 )
 (Menuitem
   (Name "Dump the database")
   (Action (SysCall `%GET_FILE_NAME_U_ALL
     %RUN_DUMP_TSA_ALL`))
)
# Thierry Le Gal  April 14, 1992
# toolkit.menu
#

# Contain the variables and the menu toolkit.menu which is appende to the desktop menu

# ----- Variables used by the menu item "Attach_note1" -----

# Displays a message on the desktop
(Variable (id BEGIN_ATTACH_NOTE1_MESSAGE)
  (Value "%.INFORM_FORM(
    % STRING(New application !!!),
    % STRING(This function: 
    - Opens a model 
    - Opens a dfd in that model 
    - Creates a text note using a flat file 
    - Attaches it to the dfd)
  )")
)

# Displays a message on the desktop
(Variable (id END.Attach NOTE1_MESSAGE)
  (Value "%.INFORM_FORM(
    % STRING(End of this application !!!),
    % STRING(The text note has been created, you can go and check it)
  )")
)

# Initialize the variables
(Variable (id MODEL_NAME) (Value "")
(Variable (id DFD_NAME) (Value "")
(Variable (id NOTE_TITLE) (Value "")
(Variable (id NOTE_PATH) (Value "")

# Get the name of the model to be opened
(Variable (id GET.MODEL_NAME)
  (Value "%.ASSIGN\%MODEL_NAME%.ENTER_FORM(
    % STRING(Input from the user),
    % STRING(Enter the name of the model to be opened in Teamwork),
    % STRING(f_Cruise_Control)
  )")
)

# Get the name of the dfd to be opened
(Variable (id GET.DFD_NAME)
  (Value "%.ASSIGN\%DFD_NAME%.ENTER_FORM(
    % STRING(Input from the user),
    % STRING(Enter the name of the dfd to be opened in Teamwork),
    % STRING(0)
  )")
)

# For debugging purpose
(Variable (id Verbose)
  (Value ".TRUE")
)

# Get the title of the note to be attached
# Define variables for note title and path

```plaintext
(Variable (id GET_NOTE_TITLE)
  (Value *%ASSIGN(%NOTE_TITLE,%t.ENTER_FORM(%STRING(input from the user),
    %STRING(Enter the title of the note to be created),
    %STRING(1234567890123456789012345678901234567890)))*)
)

# Get the path of the file to be loaded in the note

(Variable (id GET_NOTE_PATH)
  (Value *%ASSIGN(%NOTE_PATH,%t.ENTER_FORM(%STRING(input from the user),
    %STRING(Enter the path of the file to be loaded in the note),
    %STRING(/u/legal/twk/access/notes/note_text)))*)
)

# Run the executable to attach the note text using Teamwork access

(Variable (id RUN_ATTACH_NOTE1(MODEL_NAME,DFD_NAME,NOTE_TITLE,NOTE_PATH))
  (Value *%RUN(/u/legal/twk/com/attach_note1 %MODEL_NAME %DFD_NAME %NOTE_TITLE %NOTE_PATH*))
)

#----------------- Variables used by the menu item "Attach_note2" ------------

(Variable (id BEGIN_ATTACH_NOTE2 MESSAGE)
  (Value *%INFORM_FORM(%STRING(New application !!!),
    %STRING(This application attaches a note to an object
    - An Xwindow is opened from Teamwork in the com directory
    - The user is prompted to type the command 'attach_note2'
    - The program indicates what are the possible arguments for
      this function
    - When attach_note2 ends the user can kill the Xwindow which
      was opened in the the background
    Note that the user can go back and forth between Teamwork desktop
    and the Xwindow))*)
)

# Call the batch file which runs /u/legal/twk/com/attach_note2

(Variable (id RUN_ATTACH_NOTE2)
  (Value *%Bin/bash /u/legal/twk/access/batch/attach_note2.batch*)
)

#----------------- Variables used by the menu item "Attach_note3" ------------

(Variable (id BEGIN_ATTACH_NOTE3 MESSAGE)
  (Value *%INFORM_FORM(%STRING(New application !!!),
    %STRING(This application attaches a note to an object
    Note that the user can go back and forth between Teamwork desktop
    and any X-window opened by selecting it with the mouth))*)
)

# Call the batch file which runs /u/legal/twk/com/attach_note2

(Variable (Id RUN_ATTACH_NOTE3)
  (Value "/bin/ksh /u/legal/twk/access/batch/attach_note3.batch")
)

#--------- Variables used by the menu items "Open Xwindow" ---------

# Open an Xwindow, set its size (108x10) and the position of its upper left corner (0,0)
# When the Xwindow is opened, the cursor is only active in the new Xwindow. The Xwindow needs to be killed
# (<ctrl D>) so that the cursor become active again on the Teamwork desktop.
( Variable
  (Id SHELL1)
  (Value "/usr/bin/X11/aixterm -geometry 108x10+0+0")
)

# Open an Xwindow in the background, set its size (23x49) and the position of its upper left corner (1025,0)
# The fact that this Xwindow is opened in the background allow the cursor to be active in both the Xwindow
# and the Teamwork desktop. The user can go independantly in Teamwork or the Xwindow.
( Variable
  (Id SHELL2)
  (Value "/usr/bin/X11/aixterm -geometry 23x49+1025+0 &")
)

# Initialize the variables
(Variable (Id FILENAME) (Value "")

# Get the path of the name of a file
(Variable (Id GET_FILENAME)
  (Value "%.ASSIGN(%FILENAME,%.ENTER_FORM(
    %.STRING(Input from the user),
    %.STRING(Enter the path of the file to be loaded),
    %.STRING(/u/legal/twk/access/notes/notes_text)
  )")
)

# Open an Xwindow and read a file
( Variable
  (Id READ(A_FILE))
  (Value "%.SHELL1 -e /usr/bin/view %A_FILE")
)

# Open an Xwindow in the background and edit a file
( Variable
  (Id EDIT(A_FILE))
  (Value "%.SHELL2 -e /usr/bin/vi %A_FILE")
)

#--------- Menu definition ---------------------------------------------

# Provided functionalities
#---------------------------------------------------------------
# - Open_Xwindow allows you to open an Xwindow from Teamwork
# - Open_Xwindow_background allows you to open an Xwindow in the background from Teamwork
# - Open_window + read_file allows you to read a file in an Xwindow opened by Teamwork
# - Open_window + edit_file allows you to edit a file in an Xwindow opened by Teamwork
# - Attach_note1 attaches a note to a dtd. This application emphasizes the use of inputs forms to enter
# some values in Teamwork. This is my first application realizing the connection between T/User Menus
# and T/Access.
# - Attach_note2 attaches a note to the following objects: model, dde, dtd, erd, ps, matrix, sc, std, asg
#---------------------------------------------------------------
(Menu
  (Name "Tl_toolkit")
  (Menuitem
    (Name "Open_Xwindow")
    (Action (SysCall "%SHELL1"))
  )
  (Menuitem
    (Name "Open_Xwindow_background")
    (Action (SysCall "%SHELL2"))
  )
  (Menuitem
    (Name "Open_window + Read file")
    (Action (SysCall "%GET_FILENAME
      %READ(%FILENAME)"))
  )
  (Menuitem
    (Name "Open_window + Edit_file")
    (Action (SysCall "%GET_FILENAME
      %EDIT(%FILENAME)"))
  )
  (Menuitem
    (Name "Attach_note1")
    (Action (SysCall "%BEGIN_ATTACH_NOTE1_MESSAGE %GET_MODEL_NAME %GET_DFD_NAME %GET_NOTE_TITLE %GET_NOTE_PATH %RUN_ATTACH_NOTE1(%MODEL_NAME,%DFD_NAME,%NOTE_TITLE,%NOTE_PATH) %END_ATTACH_NOTE1_MESSAGE"))
  )
  (Menuitem
    (Name "Attach_note2")
    (Action (SysCall "%BEGIN_ATTACH_NOTE2_MESSAGE %RUN_ATTACH_NOTE2"))
  )
  (Menuitem
    (Name "Attach_note3")
    (Action (SysCall "%BEGIN_ATTACH_NOTE3_MESSAGE %RUN_ATTACH_NOTE3"))
  )
)
# Top level menu for including menus in the Teamwork desktop

(Include "%/CADRE_DIR/menus/general/twk_base.men")
(Include "/u/legal/twk/menus/desktop/toolkit.menu")
(Include "/u/legal/twk/menus/desktop/utility.menu")
(Include "/u/legal/twk/menus/desktop/rev.menu")
# Thierry Le Gal | April 14, 1992
# crev.menu
# Contains the variables and the menu crev.menu which is appended to the desktop menu

#---------- Variables used by the menu items * Open Xwindow *----------

# Change the directory
(Variable (Id CHANGE_DIR_CTREE)
  (Value "cd /u/legal/tkw/calltree/ctree")
)

# Opens an X window in the background in the /u/legal/tkw/calltree/ctree directory
(Variable (Id OPEN_XWINDOW_CTREE)
  (Value "/usr/bin/X11/aixterm -geometry 108x10+0+0 -T Call_tree_directory -fg Wheat -bg MidnightBlue &")
)

# Change the directory
(Variable (Id CHANGE_DIR_SOURCE_CODE)
  (Value "cd /u/legal/tkw/calltree/source")
)

# Opens an X window in the background in the /u/legal/tkw/calltree/source directory
(Variable (Id OPEN_XWINDOW_SOURCE_CODE)
  (Value "/usr/bin/X11/aixterm -geometry 108x10+0+0 -T Source_code_directory -fg Wheat -bg MidnightBlue &")
)

#---------- Menu definition -------------------

(Menu
  (Name "T1_Create_SC")
  (MenuItem
    (Name "Window_ctree")
    (Action (SysCall "%CHANGE_DIR_CTREE
                   %OPEN_XWINDOW_CTREE")

    (MenuItem
      (Name "Window_source_code")
      (Action (SysCall "%CHANGE_DIR_SOURCE_CODE
                     %OPEN_XWINDOW_SOURCE_CODE")
    )
  )
)
Appendix E

This appendix presents the programs used by the parsing tool. A description of the purpose of each function, the inputs and the outputs is given at the beginning of each program.

Files presented:

- makefile
- del_first_line.c
- count_file_lines.c
- append_to_file.c
- xlyza
- lyza.awk
- run_parse_child_list
- check_module_nature
- run_fd_children
- run_fd_parents
- xparms
- lyzout.awk
- get_input.sed
- fd_children.awk
- fd_module_kind.awk
- fd_parents.awk
- lyzparm.sed
- lyzskel.sed
- join.sed
- lyzparm.awk
# Example to create the executable count: make count
CC = /bin/cc
CFLAGS = -g
LDFLAGS = -x -g

.OBJMODS = count_file_lines.o del_first_line.o append_to_file.o

count: count_file_lines.o
del_line: del_first_line.o
append: append_to_file.o

$(CC) $(CFLAGS) -o count_file_lines.o
$(CC) $(LDFLAGS) -o del_line del_first_line.o
$(CC) $(LDFLAGS) -o append append_to_file.o
/* ...
/* del_first_line.c
/* Delete the first line of the file given as input
/*
/* Thierry Le Gal
June 2, 1992
/*
/* ...
*/

/* argv[1] is the name of the file to be processed */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <stdio.h>

#define NAME_LEN 40
#define LINE_LEN 120

main (argc, argv)
int argc;
char *argv[];
{
  FILE *fp_initial, *p_tmp;
  char *file_name, *line;
  char tmp_file_name[strlen(argv[0]) + "tmp_file"];
  char command[LINE_LEN];

  file_name = (char *)malloc(NAME_LEN);
  file_name = argv[1];

  /* Open the initial file in read mode */
  if ((fp_initial = fopen(file_name,"r")) == NULL) {
    perror("Error from fopen in del_first_line.c");
    return(-1);
  }

  /* Open the temp file in write mode */
  if ((fp_temp = fopen(tmp_file_name,"w")) == NULL) {
    perror("Error from fopen in count_file_lines.c");
    return(-1);
  }

  line = (char *)malloc(LINE_LEN);

  /* Read the first line */
  fgets(line, LINE_LEN,fp_initial);

  /* Scan through the other lines of the file */
  while (fgets(line,LINE_LEN,fp_initial) != NULL) {
    fprintf(fp_temp,"%s",line); /* Copy the initial line in the temp file */
  }

  /* Close the two files */
  fclose(fp_temp);
  fclose(fp_initial);

  /* Flush and overwrite on the initial file */
  if ((fp_initial = fopen(file_name,"w")) == NULL) {
    perror("Error from fopen in count_file_lines.c");
    return(-1);
  }

  /* Open the temp file in read mode */
  if ((fp_temp = fopen(tmp_file_name,"r")) == NULL) {
    perror("Error from fopen in count_file_lines.c");
    return(-1);
  }

  return 0;
}
pererror("Error from fopen in count_file_lines.c");
return(-1);
}

/* Copy the temp file on the initial file */
while (fgets(line, LINE_LEN, fp_tmp) != NULL) {
  fprintf(fp_initial, "%s", line);
}

/* Close the two files */
fclose(fp_tmp);
fclose(fp_initial);

/* Remove the temp file */
strcpy(command, "rm tmp_file");
if (system(command) != 0) {
  pererror("Error from rm tmp_file in count_file_lines.c");
  return(-1);
}

free(line);
free(file_name);
}
/* count_file_lines.c */
/* Counts the number of lines in the file given as an input. Insert the result at the top of this file */
/* Thierry Le Gal June 1, 1992 */

/* argv[1] is the name of the file to be processed */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <stdio.h>

#define NAME_LEN 40
#define LINE_LEN 120

main (argc, argv)
int argc;
char *argv[];
{
    int count = 0;
    int number_of_lines; /* Number of lines in the file */
    FILE *fp_initial, *fp_tmp;
    char *file_name, *line;
    char tmp_file_name[40]="tmp_file.c",
    char command[LINE_LEN];

    file_name = (char *)malloc(NAME_LEN);
    file_name = argv[1];

    /* Open the initial file in read mode */
    if ((fp_initial = fopen(file_name,"r")) == NULL) {
        perror("Error from fopen in count_file_lines.c");
        return(-1);
    }

    /* Open the temp file in write mode */
    if ((fp_tmp = fopen(tmp_file_name,"w")) == NULL) {
        perror("Error from fopen in count_file_lines.c");
        return(-1);
    }

    line = (char *)malloc(LINE_LEN);

    /* Scan through the file (also count the white lines) */
    while (fgets(line,LINE_LEN,fp_initial) != NULL) {
        fprintf(fp_tmp,"%s",line); /* Copy the initial file in the temp file */
        count++;
    } number_of_lines = count;

    /* Close the two files */
    fclose(fp_tmp);
    fclose(fp_initial);

    /* Flush and overwrite on the initial file */
    if ((fp_initial = fopen(file_name,"w")) == NULL) {
        perror("Error from fopen in count_file_lines.c");
        return(-1);
    }

    /* Store the number of lines in the first line of the initial file */
fprintf(fp_initial,"%d\n",number_of_lines);

/* Close the initial file */
tclose(fp_initial);

/* Append the temp file to the initial file */
if ((fp_initial = fopen(file_name,"a")) == NULL) {
    perror("Error from fopen in count_file_lines.c");
    return(-1);
}

/* Open the temp file in read mode */
if ((fp_tmp = fopen(tmp_file_name,"r")) == NULL) {
    perror("Error from fopen in count_file_lines.c");
    return(-1);
}

/* Scan through the file (also count the white lines) */
while (fgets(line,LINE_LEN,fp_tmp) != NULL) {
    fprintf(fp_initial,"%s",line); /* Copy the temp file in the initial one */
}

/* Close the two files */
tclose(fp_tmp);
tclose(fp_initial);

/* Remove the temp_file */
strncpy(command,"rm tmp_file");
if (system(command) != 0) {
    perror("Error from rm tmp_file in count_file_lines.c");
    return(-1);
}

free(line);
free(file_name);
/* append_to_file.c */
/* Append a last line to a file as input */
/* */
/* Thierry Le Gal June 3, 1992 */
/* */

/* argv[1] is the value to be appended at the last line */
/* argv[2] is the name of the file to be processed */

#include <sys/types.h>
#include <sys/stat.h>
#include <sys/file.h>
#include <stdio.h>

#define NAME_LEN 40
#define LINE_LEN 120

main (argc, argv)
int argc;
char *argv[];
{
    FILE *fp;
    char *file_name, *line;
    char command[LIE_LEN];

    file_name = (char *)malloc(NAME_LEN);
    file_name = argv[2];

    /* Open the file in append mode */
    if ((fp = fopen(file_name,"a")) == NULL) {
        perror("Error from fopen in append_to_file.c");
        return(-1);
    }

    line = (char *)malloc(LINE_LEN);
    line = argv[1];

    /* Append the buffer value at the end of the file */
    printf(fp,"%sn",line);

    /* Close the file */
    fclose(fp);
}
# Shell script to be run to start the extraction process for the database load
# lyza.awk creates the file .data/raw.data which should be used as input for the C routines to load the database
# Go into xparams to set the path for the source code location or indicate the full path in the makefile
#call_tree_sed="\"vt/these/ctree_appl2\""
call_tree="\"vt/these/ctree_appl2\"
#call_tree_sed="\"\"\"/demo/bspline/execsv/ctree\""
#call_tree="\"\"\"/demo/bspline/execsv/ctree\"

# flush raw.data file
echo " " > /demo/raw.data

# rm the first line in parent_list
sed -f /u/legal/twk/calltree/awk/update_child_list.sed /demo/raw.data > tmp
cat tmp > /demo/raw.data
rm tmp

# Set the value of CALL_TREE in lyza.awk
sed 's/CALL_TREE/$call_tree_sed/g' lyza.awk > tmp_lyza.awk

# Run the parser
# awk -f tmp_lyza.awk $call_tree > /demo/lyza.junk.out
awk -f tmp_lyza.awk $call_tree
# rm /demo/lyza.junk.out
rm tmp_lyza.awk
# Thierry Le Gal  June 4, 1992
# lyza.awk
#
# AWK file to analyze the calling tree file produced from CREV
# Is invoked by lyza
#
# For each module line definition, print it, extract the module name.
# The module name is the second identifier minus the final ".".

module_name = substr($2,1,length($2)-1);
$1 == 1 {
   caller_name = module_name;
   # print("%s is called by %s\n",modname,caller_name);
   print("%s \n",module_name) >> "/demo/raw.data";
}

# find comma
field = $3;
Fldnum = 3;
while ((field !~ /\$/) & (Fldnum <= NF)) {
   field = $++Fldnum;
}

# if comma was found, separate everything from field 3 to comma into
# the descriv variable
if (Fldnum <= NF) {
   descriv = $3;
   for (i=4; i<=Fldnum; i++) descriv = descriv " ";
}

# If parenthesis are in descriv then this is a function, otherwise a global variable
if ((i = index(descriv,\"(\")) == 0) {
   type = "glo";
   vtype = substr(descriv,1,length(descriv)-1);
   # print(" global var of type %s\n",vtype);
   printf("%d %s\n",3,vtype) >> "/demo/raw.data";
}
else {
   type = "fn";
   ftype = substr(descriv,1,i-1);
   # print(" lcx of type %s\n",ftype);
   printf("%d %s\n",0,ftype) >> "/demo/raw.data";
}

# Get the file name and line number from the final field having the format <./nits/showtime.c 121>.
line_num = substr($NF,1,length($NF)-1);
prew = NF-1;
filename = substr($prev,2,length($prev)-1);
# printf(" located in %s at line %s\n",filename,line_num);
print("%s %s\n",filename,line_num) >> "/demo/raw.data";

# Set to 0 the param_count value if the module defines a global var
if (type == "glo") {
   printf("%d\n",0) >> "/demo/raw.data";
}

# Run through shell command to get parameters
if (type == "fn") {
   parmcmd = "ksh xparms * filename * " line_num;
   ret = system(parmcmd);
}

# Count the number of parameters and store them in raw.data
system("cat parm.output >> /demo/raw.data");
system("rm parm.output");
}

# Find the modules invoked by the current module and store them in the child_list file
command = "ksh run_fd_children CALL_TREE " module_name;
system(command);

# Generate the global_var_invoked_list, fct_invoked_list, io_invoked_list
# and lib_invoked_list files by scanning the child_list file
command = "run_parse_child_list CALL_TREE";
system(command);

# Count the number of lines in these files and store it in the first line
command = "/c_code/count global_var_invoked_list";
system(command);
command = "/c_code/count fct_invoked_list";
system(command);
command = "/c_code/count lib_invoked_list";
system(command);
command = "/c_code/count io_invoked_list";
system(command);

# Append these files to raw.data
command = "cat global_var_invoked_list >> /demo/raw.data";
system(command);
command = "cat fct_invoked_list >> /demo/raw.data";
system(command);
command = "cat lib_invoked_list >> /demo/raw.data";
system(command);
command = "cat io_invoked_list >> /demo/raw.data";
system(command);

# Remove these files and child_list
command = "rm global_var_invoked_list";
system(command);
command = "rm fct_invoked_list";
system(command);
command = "rm lib_invoked_list";
system(command);
command = "rm io_invoked_list";
system(command);
command = "rm child_list";
system(command);

# Find the callers of the current module and store them in parent_list file
command = "ksh run_fd_parents CALL_TREE " module_name;
system(command);

# Count the number of lines in parent_list and store it in the first line
command = "/c_code/count parent_list";
system(command);

# Append the file parent_list to raw.data
command = "cat parent_list >> /demo/raw.data";
system(command);

# Remove the file parent_list
command = "rm parent_list";
system(command);

# End of if $1 == 1
# Thierry Le Gal     June 4, 1992
# $1 is the call tree name

call_tree=$1
flat_file="child_list"

# go into the right directory
    cd /u/legal/ibm/parser

# flush global_var_invoked_list, fct_invoked_list and lib_invoked_list files
    echo "" > global_var_invoked_list
    echo "" > fct_invoked_list
    echo "" > lib_invoked_list
    echo "" > io_invoked_list

# rm the first line in those three files
    /c_code/del_line global_var_invoked_list
    /c_code/del_line fct_invoked_list
    /c_code/del_line lib_invoked_list
    /c_code/del_line io_invoked_list

# initialize current_module to enter the loop
    current_module="bidon"

# loop till the child_list is empty
    while test "$current_module"
    do

# get the first line in child_list
        current_module="sed -n 1 @get_input.sed child_list"

        if test "$current_module" != ""
           then

# Check the nature of the module by scanning the call tree
            check_module_nature $current_module $call_tree

        fi

# delete the first line in child_list
        /c_code/del_line child_list

    done
# Shell script to generate the list of the children called by a module
# Is invoked by lyza.awk
# $1 is the name (with the path) of the call tree to be trimmed
# $2 is the name of the function for which we want to find the children

call_tree=$1
module=$2

go into the right directory
cd /u/legal/ibm/parser

set the value of parent_name in fd_children.awk
sed 's/parent_name/$module/g' fd_children.awk > tmp_fd_children.awk

flush child_list file before beginning
echo " " > child_list

rm the first line in child_list
./c_code/del_line child_list

get the children for the module, store them in child_list
awk -f tmp_fd_children.awk $call_tree

rm tmp_fd_children.awk
# Shell script to generate the list of the parents of a module
# Is invoked by lyza.awk
# $1 is the name (with the path) of the call tree to be trimmed
# $2 is the name of the function for which we want to find the parents

call_tree=$1
module=$2

go into the right directory
cd /u/legal/ibm/parser

set the value of child_name in fd_parents.awk
sed 's/child_name/$module/;g' fd_parents.awk > tmp_fd_parents.awk

flush parent_list before beginning
echo " " > parent_list

rm the first line in parent_list
./c_code/del_line parent_list

get the parents for the module, store them in parent_list
awk -f tmp_fd_parents.awk $call_tree

rm tmp_fd_parents.awk
# Shell script for parameter extraction from the source code.
# Is invoked by lyza.awk
# $1 is the file name of the source code (path included).
# $2 is the line number in the source code where the fct is defined.
# Change xxx to the value of the second parameter - 1.

integer line prevline
filename=$1
line=$2
prevline=line-1
cmd=s/xxx/
cmd=$cmd$prevline
cmd=$cmd/g
sed $cmd lyzskel.sed > lyz.tmp

# sed command to change yyy to the value of the first parameter
cmd=s/yyy/
cmd=$cmd$line
cmd=$cmd/g
sed $cmd lyz.tmp > lyzparm.sed

# lyzparm.sed = extract function def string from file
# join.sed = remove new line characters from function definition
# lyzparm.awk = input ftn def, output each parm name and type
# lyzout.awk = separate parm name from parm type

sed -n -f lyzparm.sed $filename > lyz.tmp
sed -n -f join.sed lyz.tmp > lyz.tmp2
awk -f lyzparm.awk lyz.tmp2 > lyz.tmp3
awk -f lyzout.awk lyz.tmp3 > parm.output

# count is a C executable which counts the number of line in a file and store it
# at the first line of this file
./c_code/count parm.output

rn lyz.tmp
rn lyz.tmp2
rn lyz.tmp3
BEGIN {
    "BEGIN block"
    if ($NF==0) exit;
"
    "match block"
    match($NF,/[a-zA-Z]/);
"
    "print block"
    printf("RSTART is %dn",RSTART);
    fname = substr($NF,RSTART,length($NF)-RSTART+1);
"
    "ftype block"
    ftype = $1;
    for (i=2; i<NF; i++) {
        ftype = ftype " " $i;
    }
"
    "if block"
    if (RSTART > 1) {
        ftype = ftype " " substr($NF,1,RSTART-1);
    }
"
    "printf block"
    printf("%s ",fname);
    printf("%s\n",ftype);
"
    "END block"
}
# Get the first line of a file

```
1 {h
 p
 q
}
```
# Thierry Le Gal  June 4, 1992
# fd_children.awk
#
# AWK file to find the modules invoked by a module
# Is invoked by run_fd_children
# parent_name must be set at the function call

BEGIN ( pre_exit_flag=0 )
{
    get_children("parent_name");
}

function get_children(MODULE)
{
    module_name = substr($2,1,length($2)-1);
    if ($1 == 1) {
        current_caller = module_name;
    }
    if (($1 == 2) && (current_caller == MODULE)) {
        called_name = substr($2,1,length($2)-1);
        printf("%s\n", called_name) >> "child_list";
        pre_exit_flag=1;
    }
    # To exit as soon as the children are found
    if ( (pre_exit_flag == 1) && (current_caller != "parent_name")) {
        exit;
    }
}
}
# Thierry Le Gal June 4, 1992
# fd_module_kind.awk
#

# AWK file to check the nature of a given module
# Is invoked by check_module_nature
# The value of ref_module is set by check_module_nature

BEGIN { ref_module = "MODULE_NAME";
    kind = 8;
}

$1 == 1 {
    definition_module = substr($2,1,length($2)-1);
    
    if ( definition_module == ref_module ) {

        # find comma
        field = $3;
        fldnum = 3;
        while ( (field !~ /,/ && (fldnum <= NF)) ) {
            field = $++fldnum;
        }

        # if comma was found, separate everything from field 3 to comma into
        # the descr variable
        if ( fldnum <= NF ) {
            descr = $3;
            for ( i=4; i<=fldnum; i++ ) descr = descr " " $i;
        }

        # if parenthesis are in descr then this is a function, otherwise a global var
        if ( ( i = index(descr,"()")) <= 0 ) {
            kind = 3;
        }
        else {
            kind = 0;
        }
    }
}

END {
    # if the ref_module is a library (kind -- -1 )
    if ( kind == 8 ) {
        # check if the module is in the system_io file
        grepcom = "grep " ref_module " system_io";
        ret = system(grepcom);
        if ( ret == 0 ) {
            kind = 1;
        }
        else {
            kind = 2;
        }
    }
}

close("./demo/raw.data");
printf("%dn", kind) >> "tmp_module_kind";
# AWK file to get the modules immediately calling the considered child
# is invoked by run_fd_parents
# child must be set at the fct call

{ get_parents("child_name");
}

function get_parents(MODULE)
{
    module_name = substr($2,1,length($2)-1);
    if ($1 == 1) {
        potential_parent = module_name;
    }
    if ( ($1 == 2) && (module_name == MODULE) ) {
        parent = potential_parent;
        printf("%s\n", parent) >> "parent_list";
    }
}
# sed script to extract from a file the parameters for a function. This is a skeleton; 17 will be
replaced by a line number and 18 will be replaced by 17+1 before this script is used. 18 should
represent the line number of the function definition. This script holds all lines beginning at 18
in the hold space until a line containing the open bracket '(' is encountered. At that time the hold
space is printed to stdout.

1,17 d
18 {h
d}
   /
   /[H
   /[g
   p
   q
}
# Join all lines in input file, replace end-of-line chars with spaces to invoke this enter: sed -n -f join.sed [filename]
{
H
g
}
$ {s/^$/ /g
p
}
# sed script to extract from a file the parameters for a function. This is a skeleton; xxx will be replaced by a line number and yyy will be replaced by xxx+1 before this script is used. yyy should represent the line number of the function definition. This script holds all lines beginning at yyy in the hold space until a line containing the open bracket '[' is encountered. At that time the hold space is printed to stdout.

```
1,xxx d
yyy {h
d
}
/[^\n]+H
/[^\n]+g
p
q
}`
# AWK file
# is invoked by xparms
# look for first ; character. If preceded by ) then parms are inside ()...
# Parse out into array elements separated by , ... else parms follow the ;
# put into array elements separated by ;

{ 
  field_n = 1;
  while (($field_n != /;/) && (field_n <= NF)) {
    ++field_n;
  }
  
  # use match to look for ";" in same or previous field
  if ((match($field_n,\/\|\|\) != 0) || (match($field_n-1,\/\|\|\) != 0)) {
    field_1 = field_n;
    while ($field_1 != /;/) {
      --field_1;
    }
  }
  
  # if () is found then there are no parms; exit printing nothing
  if ($field_1 ~ /;/) {
    exit
  }

  parenplace = index($field_1,";");
  if (parenplace == length($field_1)) {
    parmstr = $++field_1;
  } else {
    parmstr = substr($field_1,parenplace+1,length($field_1) - parenplace);
  }

  if ($field_n != /;/) field_n--;
  field_1++;
  while (field_1 <= field_n) {
    parmstr = parmstr " " $field_1;
    field_1++;
  }

  sub(/;/,"",parmstr);
  sub("/","",parmstr);
  numparms = split(parmstr,parm,;/);
}

else {
  field_1 = field_n;
  while ($field_1 != /;/) {
    field_1--;
  }
  parenplace = index($field_1,";"");
  if (parenplace == length($field_1)) {
    parmstr = $++field_1;
  } else {
    parmstr = substr($field_1,parenplace+1,length($field_1) - parenplace);
  }

  field_1++;
  while (field_1 <= NF) {
    parmstr = parmstr " " $field_1;
    field_1++;
  }

  # remove last :
  if ((substr(parmstr,length(parmstr),1)) == ";") {
    exit
  }
}
parmstr = substr(parmstr,1,length(parmstr)-1);
}
numparms = split(parmstr,parm./
);
for (i in parm) printf("%s\n",parm[i]);
}
Vita

The author was born in 1966 in Caen, a city in the ouest of France. He obtained in 1989 his degree in Mechanical Engineering from the Ecole Nationale Supérieure d'Arts et Métiers, one of the top "Grande Ecole" in France. He did his military service as an analyst programmer at the headquarters of the French Army in Paris. He came to the United States to develop his technical expertise and give an international dimension to his career. This experience was one one of the most exciting he has ever lived.