A New Level of Electronic Design Automation - Design Flow Manager - a Software Tool Implemented Using the Object-Oriented Development Methodology.

by

Krzysztof Kobylinski

A Thesis
Submitted to the Faculty of Graduate Studies
In Partial Fulfilment of the Requirements
for the Degree of

MASTER OF SCIENCE

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION PAGE

A NEW LEVEL OF ELECTRONIC DESIGN AUTOMATION -DESIGN FLOW MANAGER- A SOFTWARE TOOL IMPLEMENTED USING THE OBJECT-ORIENTED DEVELOPMENT METHODOLOGY

BY

KRZYSZTOF KOBYLINSKI

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirements of the degree of

MASTER OF SCIENCE

Abstract

This dissertation focuses on the issues related to Electronic Design Automation and Object-Oriented software development methodology. Based on the example of a PCB Design and Analysis Flow, it shows the benefits of introducing executable design flows which capture all the steps and data communication aspects of design processes. Design flows provide a means of mapping between the theoretical design steps and the CAD tools which provide the functionality to fulfill the step tasks. The implication of the proven usability of the design flows leads to the concept of a Design Flow Manager.

The Design Flow Manager is a software product which enables organizing design flows into flow libraries and provides a means of searching the libraries for the design flow meeting to the highest extent the requirements of the design process. The object-oriented software design process of the Design Flow Manager is presented with all its steps and deliverables. The Design Flow Manager is currently being investigated at the Microelectronics and Systems Software Laboratory in the Department of Computer and Electrical Engineering at the University of Manitoba.

Acknowledgments

First of all I want to thank my beloved Andreana for her total support during this work and opening for me the new exciting dimensions of love and life.

My parents participated in this work by raising me in the atmosphere of admiration for human thinking and creativity.

Dr. Bob McLeod being my advisor used the method of throwing into deep water which gave me so much space and freedom to make choices that at this moment in my life, I am happy where I am and where I am going with my professional career. Fortunately he was not consistent with his expectations of me being a neural network learning dynamics expert. His tremendous management skills enabled financing of my research for more than three years.

Mr. Tapas Shome was undoubtedly my guide in the huge and crowded jungle of contemporary technology. His bottomless knowledge of what is happening and what to read was certainly the pillar supporting my maturing as a software engineer.

Dr. W. Pedrycz, Dr. M. Pawlak, Dr. D. Blight, Peter Czezowski, Roger Ng, Budi Rahadjo, Jackson Wong and all those who were always ready to give their feedback about the research have my deep respect.

All the colleagues, students, professors and staff members of the Department of Electrical and Computer Engineering who create great friendly atmosphere in this organization deserve warm appreciation.

Mrs. Maria Stawychny who offered hours of her time to check the grammar and style of this work is highly appreciated.

Finally, I would like to thank MICRONET, the Canadian Microelectronic Corporation, and NSERC for supporting this work.

Table of Contents

1.0 Introduction

The very high complexity and variety of present design technologies create big challenges for designers. The complexity of present engineering products creates a challenge in the form of complex design processes with highly intricate design steps. As a result of this situation the learning curve for new designers is very steep. Another flavor to the challenge is added by the design tools vendors and the variety of their design tools called Computer Aided Design tools. The mapping between the theoretical design steps for a particular technology and the CAD tools which provide a means to perform the design process is often not obvious. To make the situation even worse, some design processes may not be accomplished with the CAD tools provided by one vendor creating another challenge for the transfer of design data and deliverables between different CAD tools sequenced by a design process. This thesis addresses these challenges by proposing and implementing a Design Flow Manager. The Design Flow Manager is implemented using object oriented and modern software development practices.

2.0 CAD tools and design flows.

2.1 Design process building.

Design process building is coordinating a set of activities leading to the formulation of a specific design process flow. Initially the design process goals have to be defined. The set of process goals leads to a definition of subsequent deliverables of the design process. From the deliverables the necessary design steps may be defined. Next the mapping of those theoretical design steps into CAD tools has to be done.

In this work an example of a Printed Circuit Board (PCB) Design and Analysis Process is presented. The goal of the design process is to provide a means of capturing design system information, creating a board layout and evaluating it for signal and thermal problems. The deliverables of the process are as follows:

(i) system schema,

(ii) component and trace layouts,

(iii) signal and thermal analysis reports,

(iv) manufacturing data in the form of plotter control commands.

The theoretical design steps necessary to generate the above deliverables may be defined as follows:

- schematic capture,

- component layout,

- routing,

- signal analysis,

- thermal analysis,

- manufacturing data generation.

The CAD tools which enable the realization of the above design steps may be, for example, supplied by two vendors - Mentor Graphics Inc. and Quantic Inc. Mentor Graphics provides the following tools: Librarian, Design Architect, Package and Layout for PCB design, Thermal Analysis and Fablink for manufacturing data generation. Quantic provides the following signal analysis and support tools: Database Manager, BoardScan, Wave Probe, Signal Viewer and Greenfield.

The number of CAD tools indicates that the mapping between them and the theoretical design steps is not one-to-one which creates another challenge for the designer. The designer has to possess enough knowledge on the specific aspects and functionality of the particular CAD tools to be able to perform the design in the CAD tools domain instead of the domain of the design steps. Using CAD tools provided by two different vendors may or may not create another challenge for design data and deliverables communication. In the above case, tools interfacing is supported by the Mentor-Quantic Interface, which enables data transfer from the design into the signal analysis stage.

The function of the CAD tools and the flow of the design process is presented in the next section where an executable flow for the process is introduced.

2.2 Design process flows

Mentor Graphic Inc. introduced two tools which provide the platform and a means to create, instantiate and execute design flows. WorkXpert is a specialized graphic editor which enables the building and modifying of design flows using standard components. It enables flow capture which may be described as an activity to define the steps that build the design process, their sequence and dependencies between their actions and the results of the other step executions. Under the definition of the design steps we should understand the specification of the desired behavior which is expected during step execution. FlowXpert provides the platform and environment to manage compiled flows instantiation and execution. It monitors and controls flow execution through dependencies and state functions and also automates flows where appropriate [3].

A design flow is a compiled, self contained software structure which provides the designer with a graphical, fully functional representation of the design algorithm and may be instantiated for a specific design process case. A design flow is a system built from three main components:

- graphical template,

- set of application tools,

- set of defined executable actions.

An instance of a design flow additionally preserves specific design data and deliverables. The state of design data and deliverables correspond to a certain state in the specific design process instance.

The graphical template presents the design process in the form of a task graph with different kinds of steps and interconnections between them which define the control flow. The following steps may be applied as flow components:

- task steps,

- subflow steps,

- decision steps,

- activity steps.

Different steps correspond to different activities. Task steps enable the activation of any actions in the form of executable programs in the local operating system shell. Each of the task steps may have defined links for the pre-execute, execute and post-execute actions. This enables not only CAD tool launching but also other activities around the design process such as the automation of design data import-export between CAD tools.

Subflow steps encapsulate complete flows which correspond to the main parts of the design process and very often are characterized by their own deliverables. Subflow steps enable the introduction of a hierarchy of design steps or tasks. Using a

hierarchy of design steps provides a means to apply the human problem-solving method of decomposition of complex problems into well-defined subproblems [1].

Decision steps provide the mechanism to control and customize design process flows by providing definable decision inquiring dialog boxes.

Activity steps are very similar in their abilities and definitions to task steps. The only difference is that these are not sequenced with other steps into design flows but exist as independent activities which may be executed at any time during the design flow execution.

2.3 PCB design and analysis flow.

The Printed Circuit Board (PCB) Design and Analysis Flow has been created to automate the process of PCB design data inquiring, design capture, thermal and signal analysis, reliability testing, and manufacturing data generation.

The PCB flow is built as a hierarchical design step structure which decomposes the design and analysis process into subprocesses corresponding to specific main task groups which lead to the generation of process deliverables. The main task groups have been distinguished and as a result, the top level of the flow consists of five main components:

- Design Capture subflow,

- Signal Analysis subflow,

- Thermal Analysis subflow,

- Reliability Test step,

- Manufacturing Data step.

Additionally the flow contains a *Design Setup* step and four decision steps (Figure1).

FIGURE 1. PCB Design and Analysis flow template with design setup and decision step prompts.

The *Design Setup* step prompts the designer to enter an existing or new design name. Defining the design name value enables either a connection to existing design data and deliverables generated during past sessions of the design process instance or to create a new instance of a design process. All tools activated during the design process are linked automatically to the design instance data.

2.3.1 Design Capture subflow.

The *Design Capture* subflow gathers all the tasks and steps necessary to complete design capture with two main deliverables. These are the schema and board layout (Figure 2). The *Design Components* activity step enables the execution of the Mentor Graphics Librarian tool. This allows the designer to create a database specific for the design which organizes and stores design specific components, component mapping files and design geometries such as: logos, padstacks, component geometries, board geometry and mechanical parts specifications. The reason why this step is defined as an independent activity step is to enable the designer to introduce any database upgrades at any time during the design capture process. Three sequenced design steps are:

- *Schematic Entry,*

- *Data Preparation,*

- *Board Layout.*

The *Schematic Entry* step executes the Mentor Graphics Design Architect tool, which is a specialized graphic editor providing a means for schematic capture. Schematic capture is an activity which includes such tasks as components pick up from libraries or new hardware components definition, component connections definition, and components and connections properties specification. After the schema has been captured, the design process flow goes to the next step called *Data Preparation.* This step executes the Mentor Graphics Package tool. This tool performs mapping of logical symbols on a schema to the corresponding physical components, checking the design for design rule errors and creating support design objects like components, gates, nets and pins to be transferred into a layout tool [5].

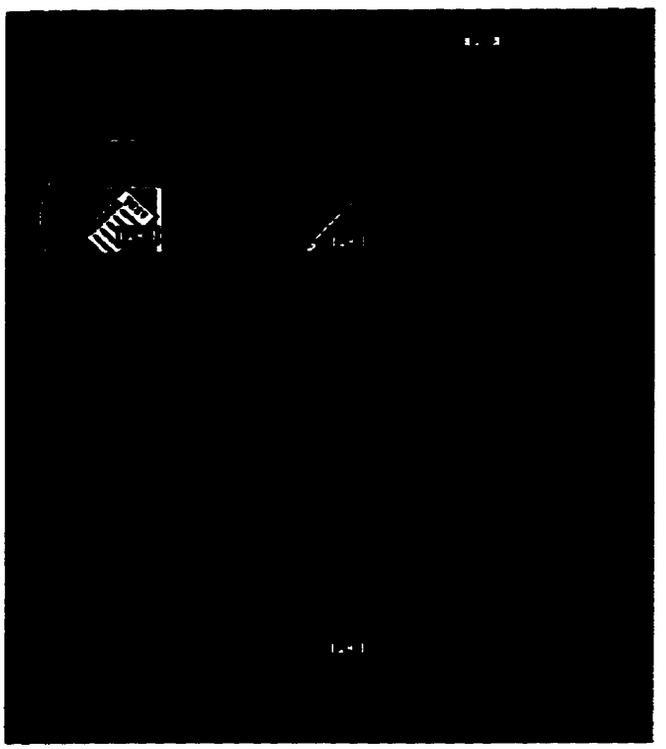


FIGURE 2. Design Capture subflow template.

The *Board Layout* step is activated next by executing the Mentor Graphics Layout tool. This tool enables the transformation of the system schema into a physical board layout. The activities performed in this tool are board layers definition, component placement and nets routing. All of them may be performed either manually or automatically.

The deliverables of the complete *Design Capture* subflow are the system schema and the board layout with its physical characteristics.

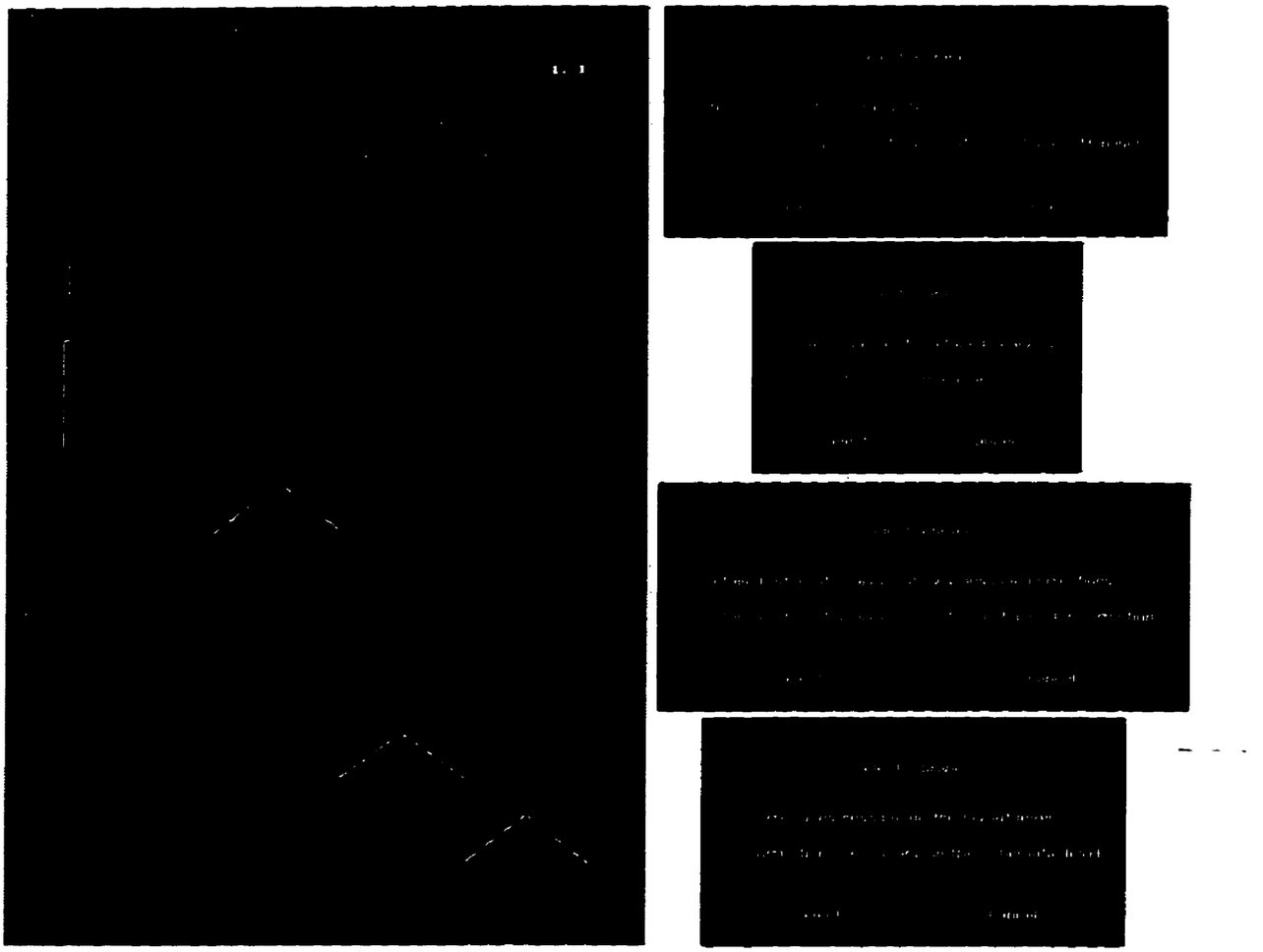


FIGURE 3. PCB Signal Analysis subflow template with decision prompts.

2.3.2 Signal Analysis subflow.

The *Signal Analysis* subflow is highly customized to accommodate the features of analysis tools which are specific to the chosen vendors. The analysis methodology is also not formalized and the features of the analysis tools leave lots of freedom for the designer to create a desired analysis methodology.

The subflow consists of six design steps which execute five different tools. The looping control mechanism provides the designer with an automated, efficient and easy to use analysis system (Figure 3). The subflow starts with the *Analysis Interface* step which executes the Mentor Graphics Layout tool and the board layout is loaded into the tool. In this case the Layout tool has activated the Quantic interface as one of its menu options. The interface enables the extraction of the design data and the generation of analysis input files required by the Quantic tools.

Next, the decision step enables the redirection of flow control to the *Components Translation* step. This step executes Quantic's Database Manager which allows substitutions for components not found in Quantic's database. The tool also provides a means of displaying, adding and deleting items in the database. Next, the flow control is redirected to the *Check Layout for Problems* step which executes Quantic's BoardScan. This tool scans the board to detect possible analog problems and as a result generates an analysis report file. The report file consists of information about critical nets and values defining the scale of specific phenomena. During the post-execute action of the step a scripting file is executed which automatically extracts the information about the number and names of critical nets from the report file and generates a critical nets listing. If the number of critical nets is equal to zero, then the run of the subflow is finished on the step *NO PROBLEMS*. In other cases subflow control goes to the step called *Signal on the Worst Net*, where Quantic's Wave Probe tool is executed. The tool displays the graphical representation of the signals on the most critical net (Figure 4). At this stage of the process, the designer's experience plays a crucial role since he has to decide if the problems found on the most critical net may be eliminated by introducing any changes on the layout or schematic level and if the elimination of the problems may introduce any improvements to the other critical nets. The *What-if Analysis on the Net?* decision box enables sending the worst net to detailed analysis or picking up the next worst net from the listing and loading it into Quantic's Signal Viewer tool.

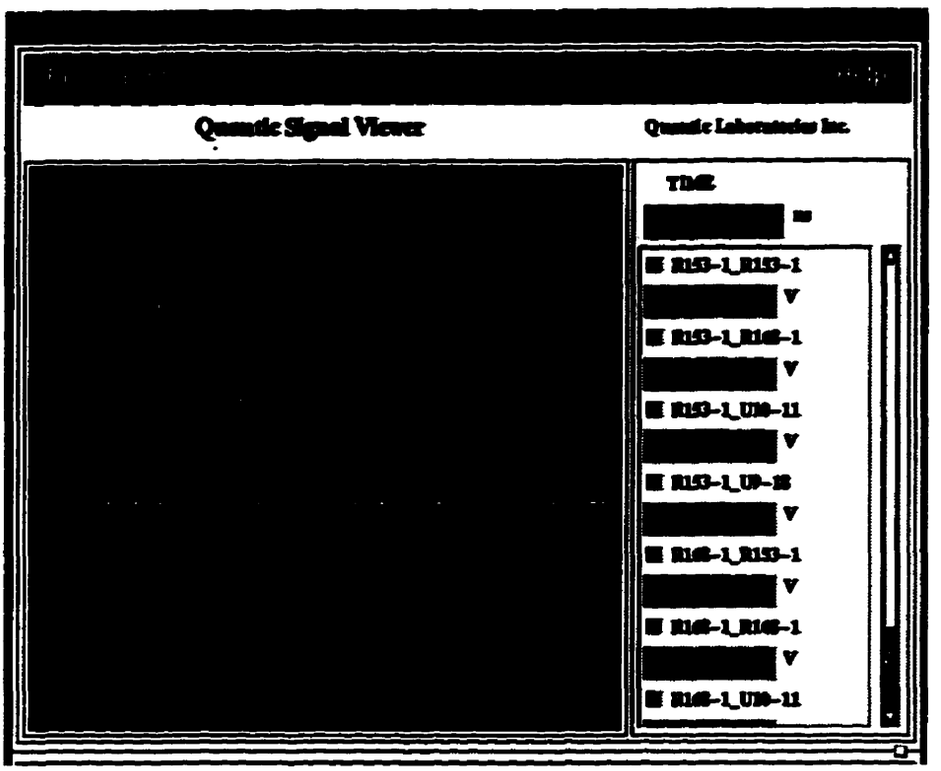


FIGURE 4. Signals on the critical net.

Detailed "what-if" analysis is performed in Quantic's Greenfield tool executed from the *What-if Analysis* step. The system loads the actual critical net to the Greenfield tool where an interactive analysis for cross-section modifications and schematics rebuilding may be performed. The tool also enables the creation or verification of design rules in terms of conductor shapes and sizes, dielectric types and sizes and placement of conductors, ground and power planes. The detailed analysis shows if there exists any possible corrections to the problems on the net and if the corrections may be performed on the layout or schematic level. In case the analysis does not provide any correct solution, the *Possible Correction* decision step redirects the flow control to the *Pick up the Next Worst Net* step. This step executes a

scripting file which extracts the next critical net from the listing generated from the BoardScan report file.

When the results of detailed analysis provide a correction solution, the flow control may be redirected to the appropriate layout or schematic tool to introduce necessary changes.

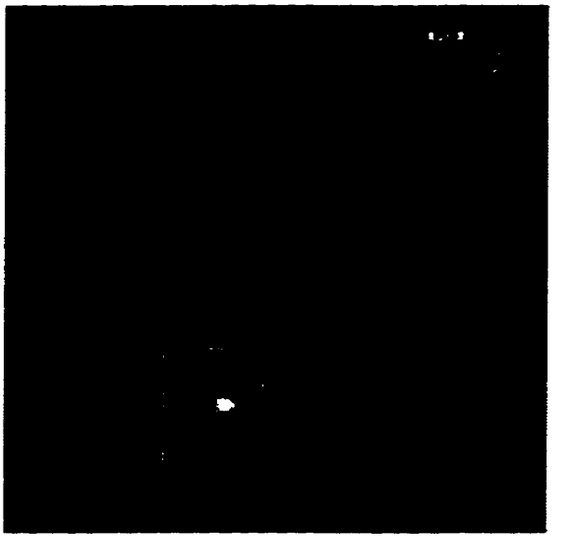


FIGURE 5. PCB Thermal Analysis subflow template.

2.3.3 Thermal Analysis subflow.

The *Thermal Analysis* subflow consists of two task steps: *Analysis Interface* and *Thermal Analysis*. The *Analysis Interface* step executes Mentor Graphics' Layout tool which enables the designer to generate an ASCII Geometries File. The ASCII Geometries File may be imported into Mentor Graphics' AutoTherm tool. The AutoTherm tool enables interactive thermal analysis of a PCB design by simulating heat transfer, analyzing thermal behavior and generating graphical results.

2.3.4 Reliability Test step.

After performing signal and/or thermal analysis, flow control is directed to the *Reliability Test* step. Mentor Graphics' Reliability Manager tool executed in this step enables the generation of predictions for electronic systems life time. To achieve reliability estimation in the form of Mean Time Between Failures (MTBF) value, the tool employs commonly used failure rate models and the results of analysis tools. The Reliability Manager directly accesses design and analysis data, computes the failure rate and graphically displays results.

2.3.5 Manufacturing Data step.

In the case that the analysis results and reliability level of the design are satisfactory, the design may be exported to manufacturing. The *Manufacturing Data* step provides all necessary features for generating manufacturing data and documentation of the printed circuit board design. The tool executed in this stage is Mentor Graphics' FabLink. The output of the FabLink tool contains photoploter (artwork) data, drilling and milling data, fabrication (detail) and assembly drawings, a bill of materials and a variety of reports [4].

2.4 Other design flows.

Another project which is currently under development in the Microelectronic and Systems Software Laboratory is a design flow which will provide a means to model the behavior of hardware systems. The flow will provide graphical design entry and enable simulation to evaluate the correctness and generation of the corresponding VHDL code. The CAD tools employed to analyze and compile the VHDL code, and to perform functional and timing simulations are supplied by Synopsys Inc. After satisfactory simulation runs, the systems will be exported to target technology for final syntheses [14].

2.5 Benefits of introducing design flows to the design environment.

Analyzing the existing design flows leads to the conclusion that the employment of design flows into design automation gives designers several advantages [3]:

- use of dependable, repeatable design processes,

- reusable process expertise based on the state of the art experience across the organization,

- automation of repetitive or complex tasks that allows engineers to concentrate on design issues,

- leverage existing tool customizing,

- hierarchical process decomposition into well-defined subproblems,

- multi-discipline, multi-framework flow integration enables coordination of tasks in a multi-vendor tool environment,

- multi-user and multi-platform real time project tracking which provides design team members with dynamic task and data status and automated and conditional notification of critical design process events,

- a decrease in the learning curve especially for new or junior designers.

2.6 Motivation to introduce a Design Flow Manager

The flexibility and benefits of the flow approach make it likely that flows will be used as the primary mechanism to automate and manage design expertise in the near future. From the perspective of any design organization we see the implementation of this approach in the form of design flow libraries. The design flow libraries will categorize flows and organize them in clusters. The flow libraries may for example group flows which support the same design technology and provide different intricacy of design steps and deliverables. Because of this an aspect of flow libraries management will be to provide mechanisms to search libraries for the required flows. This is the perfect moment to introduce the Design Flow Manager tool, which will provide a means to manage flow libraries and flows. The tool is described in the next chapter along with the complete software design process which led to its implementation.

3.0 Design Flow Manager - object-oriented software design process.

The complete software design process consists of the following major steps: Analysis, Design, Coding, Testing, Release and Maintenance [13] (Figure 6). The Design Flow Manager presented here was developed with the help of the Object Modeling Technique (OMT) which is one of the commonly used object - oriented software development methodologies [8].

FIGURE 6. Software design process.

3.1 Application requirements.

The Design Flow Manager as proposed in the previous chapter is expected to provide a means for:

- accessing the tools which enable the creating and/or rebuilding flows; in the particular case these are WorkXpert and FlowXpert from Mentor Graphics Inc.,

- customizing and managing flow libraries,

- searching flows,

- executing flows.

3.2 Analysis.

Analysis is the phase of the software design process where project requirements are being formalized and functionality and structure of the system is being analyzed. The two main groups of design activity during this stage are requirements analysis and system analysis. (Figure 7)

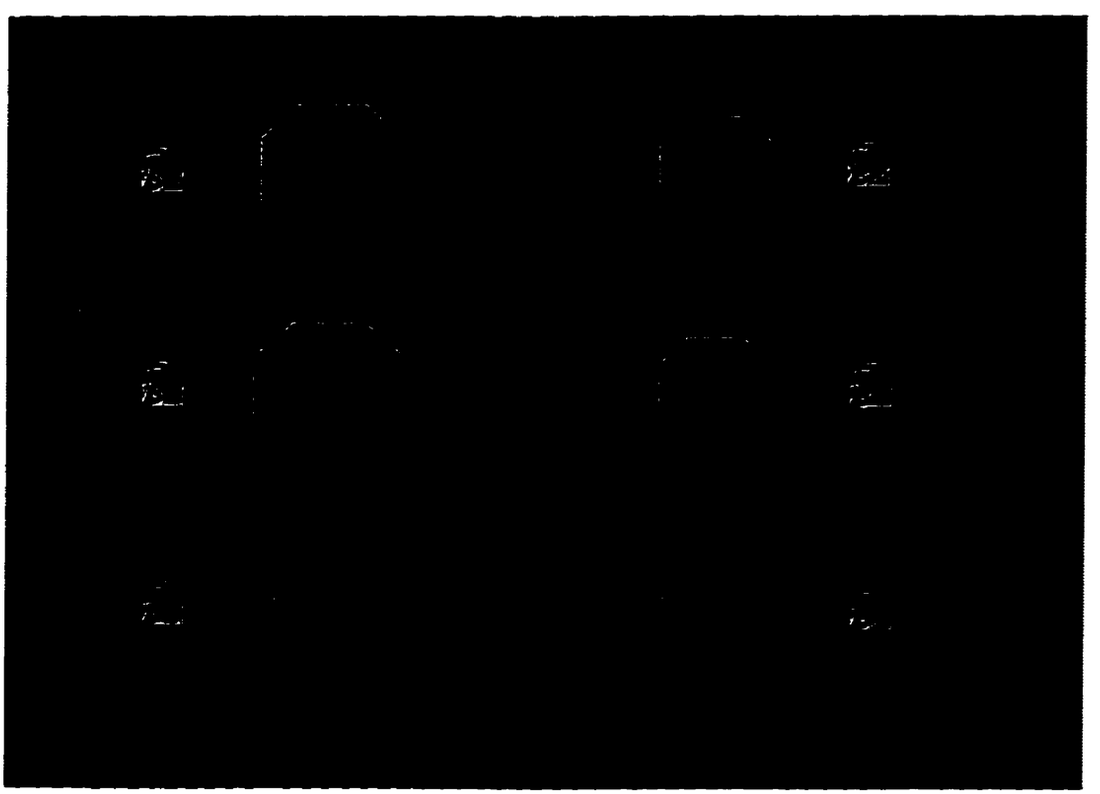


FIGURE 7. Analysis Phase.

3.2.1 Requirements analysis.

This is an extremely important phase of the process with the main focus oriented towards a deep and complete understanding of user needs and the expected func-

tionality of the system. Close cooperation between the user and the designer is necessary which very often has the form of a series of recorded sessions. As a result of this step three deliverables are generated: a Requirements Statement, a set of Use Case Scenarios, and a Graphic User Interface prototype [6].

3.2.1.1. Requirements statement.

A very good understanding of the expected system functionality and its environment is necessary to generate a Requirements Statement. Informal and often incomplete user expectations have to be refined and specified in a formal way.

The set of requirement statements for the Design Flow Manager is defined as follows:

- the tool should enable the user to organize the design flows into libraries which will have the form of file structures preserving all the information regarding localization of flows and the design steps of these flows,

- the management of flows should enable the user to add new and delete old flows from a specific library, as well as create new flow libraries and delete obsolete ones,

- based on the set of required design steps, a searching mechanism should be implemented, which will generate statistics reflecting the degree of match between the set of desired design steps and the set of design steps building up the available flows,

- the tool should enable executing both WorkXpert and FlowXpert tools for purposes of creating and executing new flows,

- only one library at a time should be activated granting access to its flows,

- the tool should enable viewing design steps of any chosen design flow,

- the tool should display graphical representation of flows in the form of icons which should enable executing them with the help of a standard pointing device,

- the tool should reserve library management tasks only for users granted administration privileges.

3.2.1.2. Use case scenarios.

Based on the analysis of user requirements a set of Use Case Scenarios have been generated. Use cases describe a system's functionality in terms of defining and describing possible scenarios of system use and interaction with its environment. The functionality of the system is divided into less complex operations which present a sequential events exchange between objects upon which the system is built. Each operation may have a number of possible scenarios which show optional action flows [6][7]. There is one Use Case Event Trace created for every Use Case Scenario. It represents a sequence of events taking place between objects during a system operation which corresponds to a particular scenario. All the communication signals between objects are considered to be events [7].

The Event Flow shows a summary of all events taking place between all the objects making up the system [6] (Figure 22).

The set of use cases extracted from the requirement statements of the Design Flow Manager defines the required functionality of the tool. The set includes the following use cases (Figure 8):

- Grant User Administration Privileges (NewAdmin),

- Revoke User Administration Privileges (DeleteAdmin),

- Create New Library,

- Delete Library,

- Add Flow,

- Update or Upgrade Flow,

- Delete Flow,

- List Existing Libraries,

- Open Library,

- New Flow,

- View Design Steps,

- Open Flow,

- Search Flow.

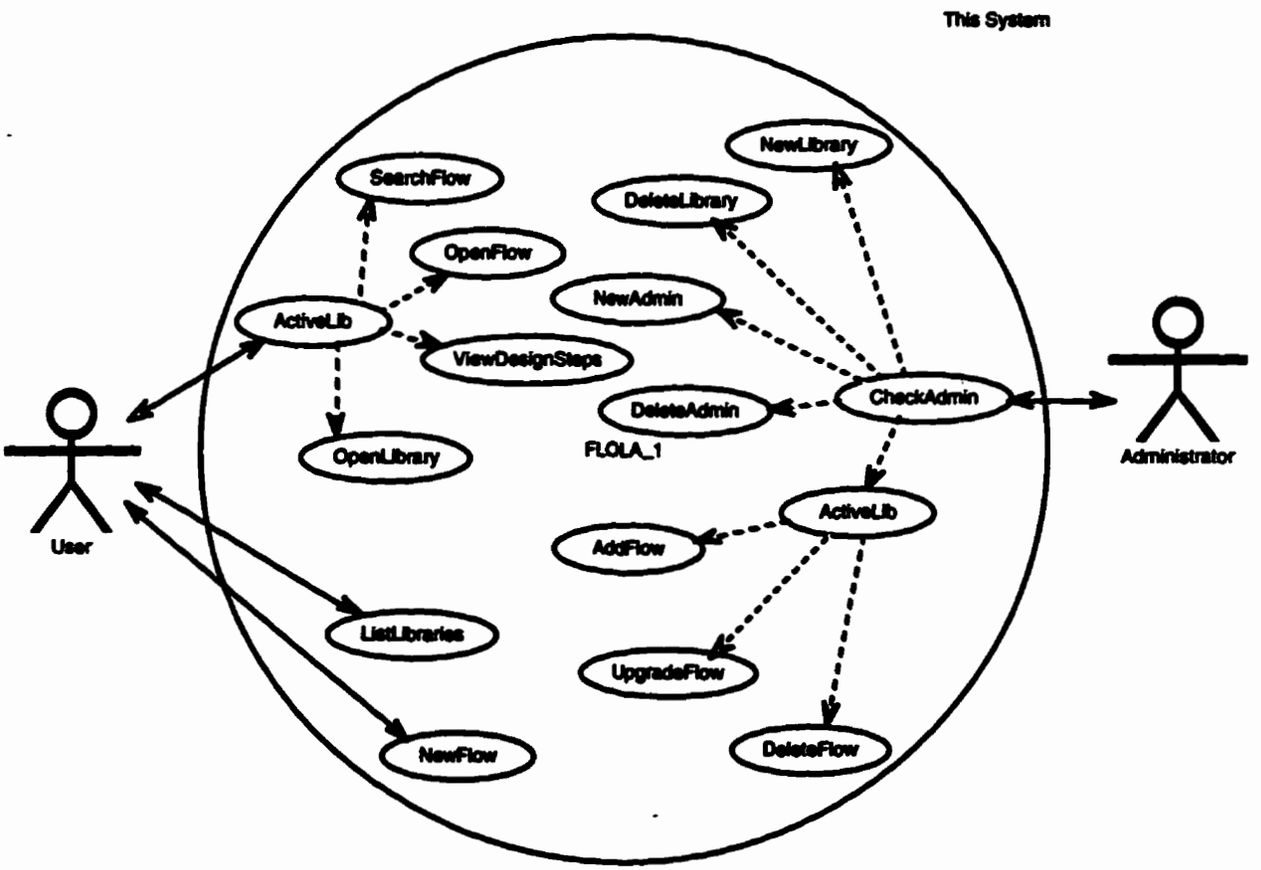


FIGURE 8. Set of Design Flow Manager use cases.

The diagram presenting the use cases set shows inheritance hierarchy (Figure 8). Some use cases inherit from the CheckAdmin and some from the ActiveLib use cases. The application of inheritance to use cases has been discussed in [7]. The CheckAdmin use case represents the system activity which evaluates whether the user is granted the administration privileges to perform the inherited operations or not. The ActiveLib use case represents the actions leading to activation of a specific flow library.

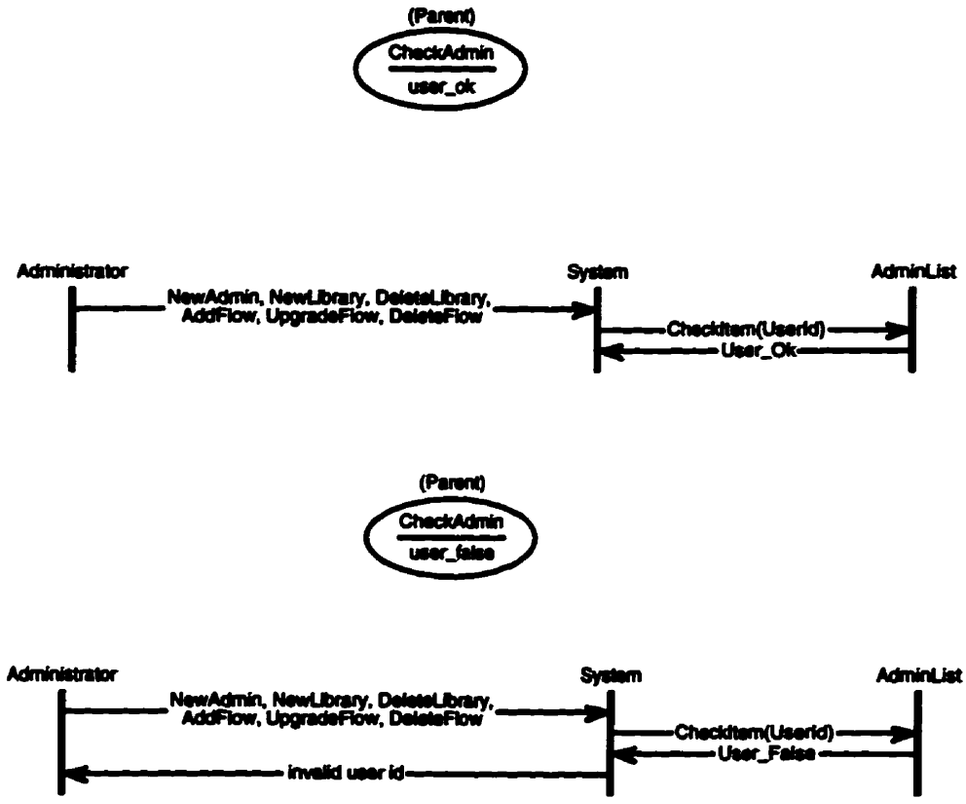


FIGURE 9. CheckAdmin use case event traces. Two scenarios: user_ok and user_false.

3.2.1.2.1. CheckAdmin use case (Figure 9).

The textual description of the system activity during the CheckAdmin use case is:

- the user activates one of the following menu options: NewAdmin, DeleteAdmin, NewLibrary, DeleteLibrary, AddFlow, UpgradeFlow or DeleteFlow,

- the system extracts from the set of environment variables the value of USER which matches userid,

- the system sends a CheckItem event with the argument of user ID to the AdminList which preserves information about users who have been granted administrative privileges,

- the user ID is verified and if it is included in the AdminList the system enables activation of any administrative tasks,

- in the case that the user ID is not included in the AdminList, the system sends a negative message to the user.

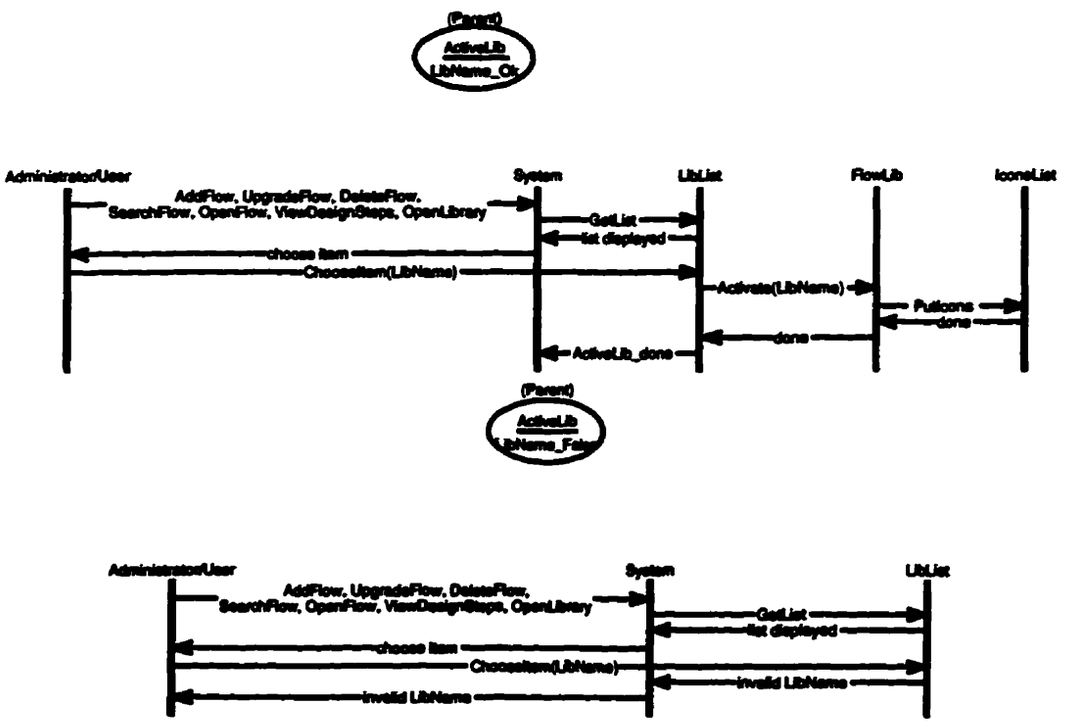


FIGURE 10. ActiveLib use case event traces. Two scenarios representing scenarios for existing and non-existing library name LibName.

3.2.1.2.2. ActiveLib use case (Figure 10).

The textual description of the actions performed during the ActiveLib use case is:

- the user activates one of the following menu options: SearchFlow, OpenFlow, ViewDesignSteps, OpenLibrary or any of the following menu options: AddFlow, UpgradeFlow or DeleteFlow activated by the control flow after successful execution of the CheckAdmin use case,

- the system sends a request to a library list called LibList to return all the elements of the list,

- the system displays all the elements of the LibList asking the user to choose one of them,

- the system activates the chosen flow library,

- the chosen library extracts from the IconList the names and locations of flow icons to be displayed in the icon window,

- in the case that the library name chosen is invalid, the system returns an error message to the user.

3.2.1.2.3. NewAdmin use case (Figure 11).

The textual description of the actions performed during the NewAdmin use case is:

- the user selects the Grant User Administration Privileges menu option,

- the system after successfully performing the CheckAdmin use case, requests the user to enter a new administrator user ID,

- the user enters the new administrator user ID,

- the system sends AddItem event to AdminList passing the user ID value as an argument,

- the AdminList object performs the procedure to attach a new element and returns control to the system,

- in the case that the user enters a user ID of an existing administrator, the system returns an error message.

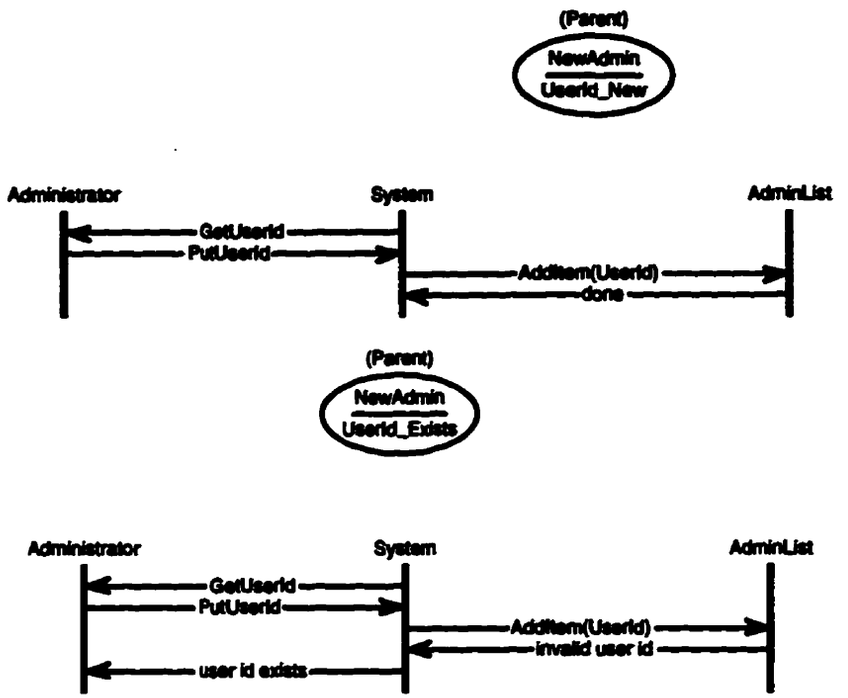


FIGURE 11. NewAdmin use case event traces. Two case scenarios for successfully adding a new administrator and a redundant run for an existing administrator.

3.2.1.2.4. DeleteAdmin use case (Figure 12).

The textual description of the actions performed during the DeleteAdmin use case is:

- the user selects the Revoke User Administration Privileges menu option,

- the system after successfully performing the CheckAdmin use case, requests the user to enter obsolete administrator user ID,

- the user enters the administrator user ID,

- the system sends a DeleteItem event to AdminList passing the user ID value as an argument,

- the AdminList object performs the procedure to detach the specified element and returns control to the system.

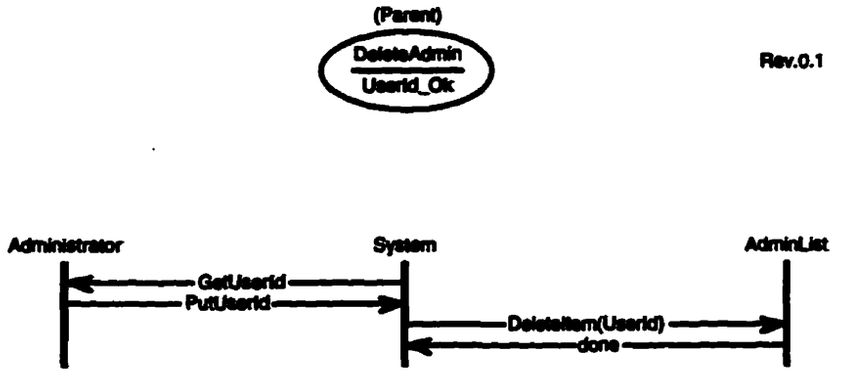


FIGURE 12. DeleteAdmin use case event trace.

3.2.1.2.5. NewLibrary use case (Figure 13).

The textual description of the actions performed during the NewLibrary use case is:

- the user selects the Create New Library menu option,

- the system after successfully performing the CheckAdmin use case, requests the user to enter a new library name,

- the user enters the new library name,

- the system sends event AddItem to the LibList object passing the LibName with the value of the library name as an argument,

- the LibList attaches the new element to the list and instantiates an object of the new library - FlowLib,

- the FlowLib object sequentially sends events to create LibStepList, FlowList and IconeList objects,

- after successful creation of all its composite objects, the FlowLib object returns the flow control to the system,

- in the case that the library name entered by the user is invalid, the LibList object does not attach a new item but returns an error message to the system which passes the message to the user.

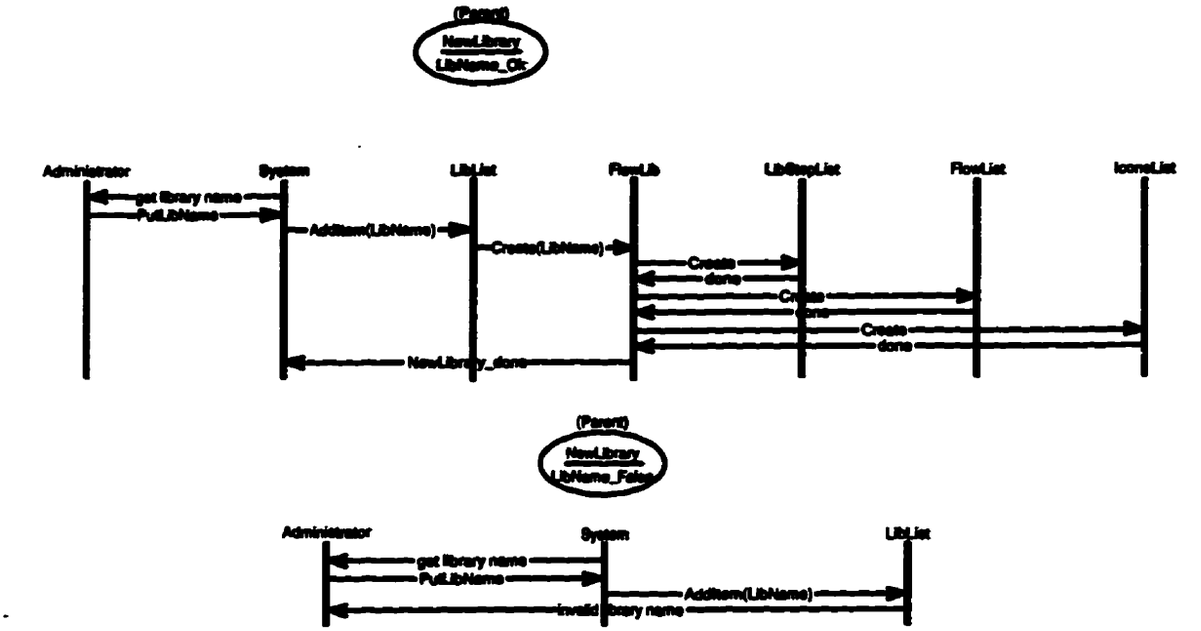


FIGURE 13. NewLibrary use case event traces. Two case scenarios for a successful run and a failure because of an invalid library name.

3.2.1.2.6. DeleteLibrary use case (Figure 14).

The textual description of the actions performed during the DeleteLibrary use case is:

- the user selects the Delete Library menu option,

- the system after successfully performing the CheckAdmin use case, requests the user to enter the obsolete library name,

- the user enters the obsolete library name,

- the system sends DeleteItem to the LibList object passing as the argument LibName which has the value of the obsolete library name,

- the LibList object detaches the item corresponding to the library name and sends a Destroy events to its composite objects: LibStepList, FlowList, IconeList,

- after successful destruction of its components, the FlowLib object returns the flow control to the system,

- in the case that the entered by the user library name is invalid, the LibList returns an error message to the system.

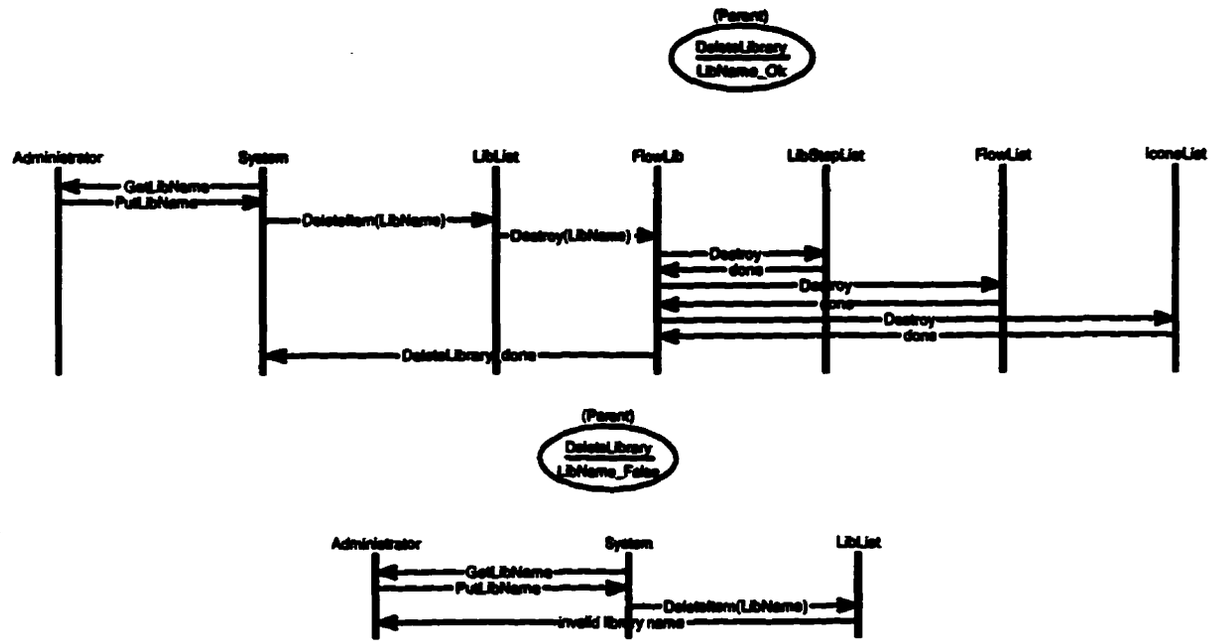


FIGURE 14. DeleteLibrary use case event traces. Two case scenarios for a successful run and a failure because of an invalid library name.

3.2.1.2.7. AddFlow use case (Figure 15).

The textual description of the actions performed during the AddFlow use case is:

- the user selects Add Flow menu option,

- the system after successfully performing the CheckAdmin and ActiveLib use cases, requests the FlowLib object to return list of existing flows,

- the FlowLib object requests the FlowList object to return the list of existing flows, which is being displayed to the user,

- the system requests the user to enter a new flow name,

- the user enters the new flow name,

- the system creates new instance of the Flow passing the FlowName value as the attribute,

- the new object of the Flow sends a request to the FlowList to be attached to the list,

- the Flow object creates an instance of FlowStepList which sends the request to the system to receive design steps of the new flow,

- the system requests the user to enter the design steps,

- the user enters the design steps,

- after receiving the design steps, the FlowStepList object attaches them to its list and sends them to the LibStepList object which stores design steps of all the flows contained by the active library,

- the LibStepList attaches the design steps to its list,

- the Flow object requests the system for the new flow icon data, and the system passes this request to the user,

- the user enters the new flow icon data,

- the Flow object receives the icon data and passes it to the IconeList,

- the IconeList object attaches the new flow icon name to its list and returns the control to the system,

- in the case that the user has entered an invalid flow name, the FlowList object sends an error message to the system after it has received the AddItem event from the Flow object. The system passes the error message to the user,

- in the case that the user has entered invalid icon data, the Flow object sends an error message to the system which passes the message to the user.

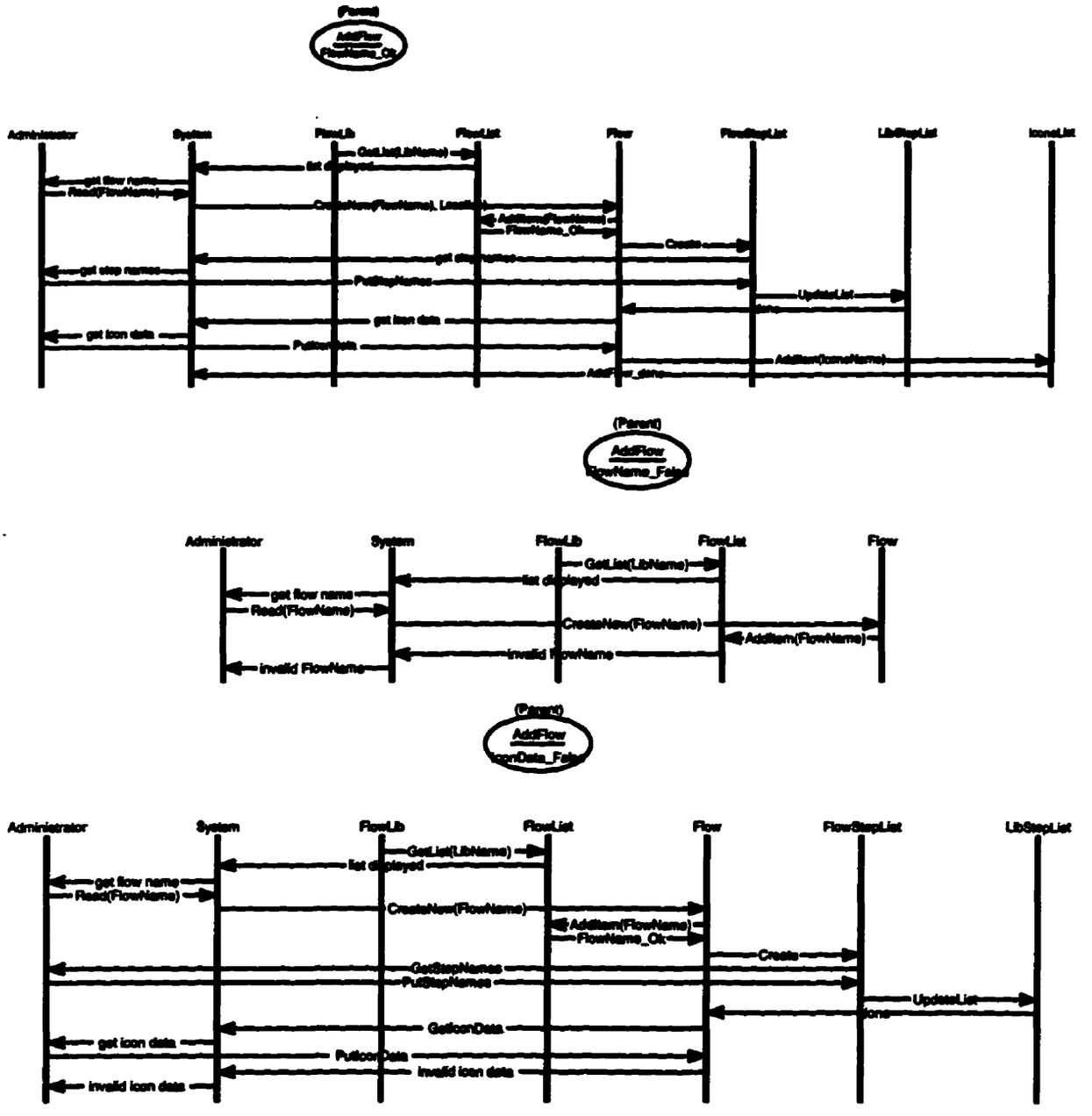


FIGURE 15. AddFlow use case event traces. Three case scenarios for a successful run and two failure runs because of an invalid library name and invalid flow icon data.

3.2.1.2.8. UpgradeFlow use case (Figure 16).

The textual description of the actions performed during the UpgradeFlow use case is:

- the user selects the Upgrade Flow menu option,

- the system after successfully performing the CheckAdmin and ActiveLib use cases, requests the FlowLib object to return the list of existing flows,

- the FlowLib object passes the request to the FlowList object,

- the FlowList object returns to the system the list of flows contained in the active library which is displayed to the user,

- the user is requested to enter the flow name,

- the user enters the flow name, which causes the system to send the Edit event to the Flow object,

- the flow is edited and the user has the opportunity to introduce changes to the flow structure,

- upon completion of the flow editing, the Flow object sends UpdateList events to both FlowStepList and LibStepList, requesting both objects to introduce changes in terms of design steps,

- upon completion of the step lists update, the Flow object returns the control to the system.

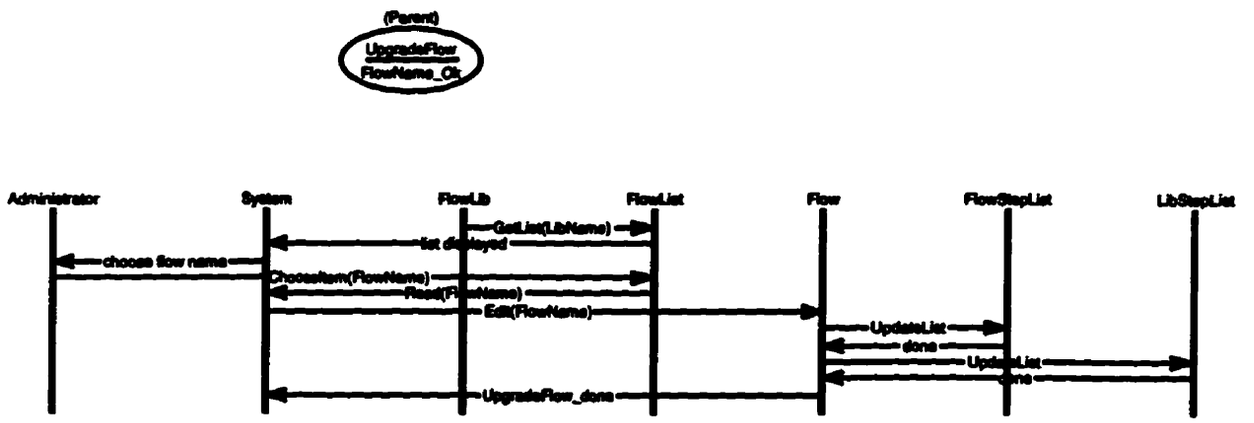


FIGURE 16. UpgradeFlow use case event trace.

3.2.1.2.9. DeleteFlow use case (Figure 17).

The textual description of the actions performed during the DeleteFlow use case is:

- the user selects the Delete Flow menu option,

- the system after successfully performing the CheckAdmin and ActiveLib use cases, requests the FlowLib object to return the list of existing flows,

- the FlowLib passes the request to the FlowList object,

- the FlowList object returns to the system the list of flows contained by the active library which is displayed to the user,

- the user is requested to enter the flow name,

- the user enters the flow name, which causes the system to send a DeleteItem event to the FlowList object,

- the FlowList object detaches the item from its list and sends a Destroy event to the Flow object,

- the Flow object sends a Destroy event to the FlowStepList object, which asks the LibStepList object to DeleteItems from its list,

- the Flow object sends a DeleteItem event to the IconList passing the flow name as the argument,

- the IconList object detaches the item corresponding to the flow icon data and returns the control flow to the system.

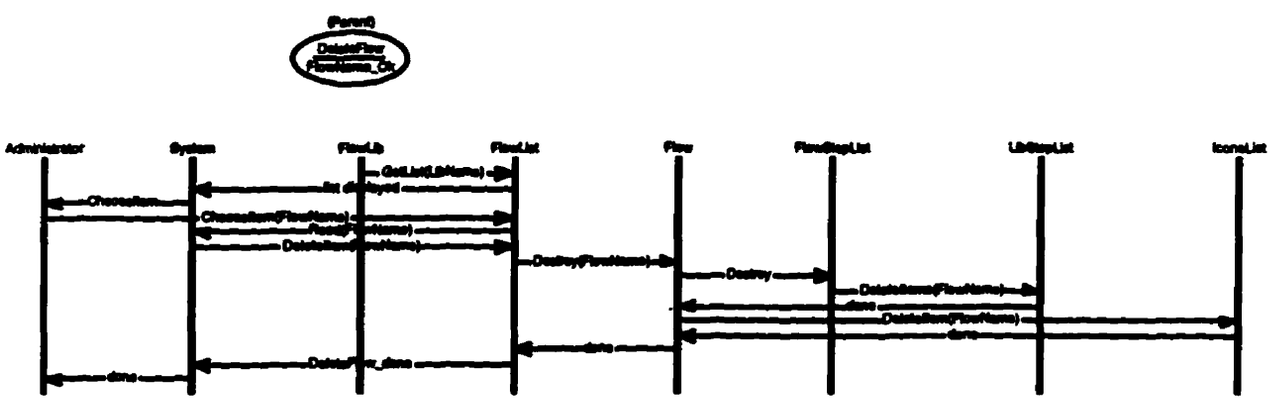


FIGURE 17. DeleteFlow use case event trace.

3.2.1.2.10. ListLibraries use case (Figure 18).

The textual description of the actions performed during the ListLibraries use case is:

- the user selects the List Existing Libraries menu option,

- the system sends a GetList request to the LibList object,

- the LibList object returns the list of libraries to the system which displays them to the user.

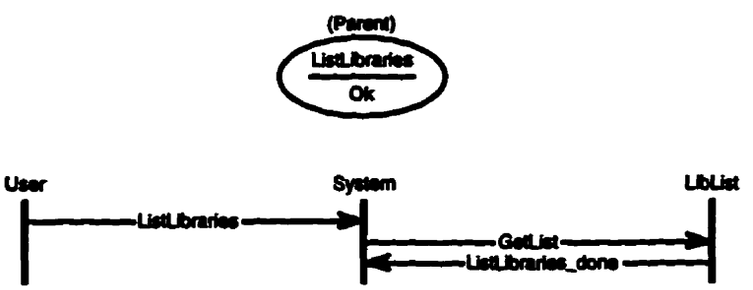


FIGURE 18. ListLibraries use case event trace.

3.2.1.2.11. OpenLibrary use case.

This use case is just an instance of the ActiveLib use case.

3.2.1.2.12. NewFlow use case.

The textual description of the actions performed during the NewFlow use case is:

- the user chooses the NewFlow menu option,

- the system executes the WorkXpert to enable the user to work on a new flow.

3.2.1.2.13. ViewDesignSteps use case (Figure 19).

The textual description of the actions performed during the ViewDesignSteps use case is:

- the user chooses the ViewDesignSteps menu option,

- the system after successfully performing the ActiveLib use case, requests the FlowLib object to return the list of existing flows,

- the FlowLib object requests the FlowList object to return the list of flows contained in the activated library,

- the system displays the list of flows,

- the system asks the user to choose the flow in order to view its design steps,

- the user chooses the flow,

- the system sends a GetList event to the FlowStepList object which is a part of the chosen Flow object,

- the FlowStepList object sends its list of steps to the system,

- the system displays the design steps to the user.

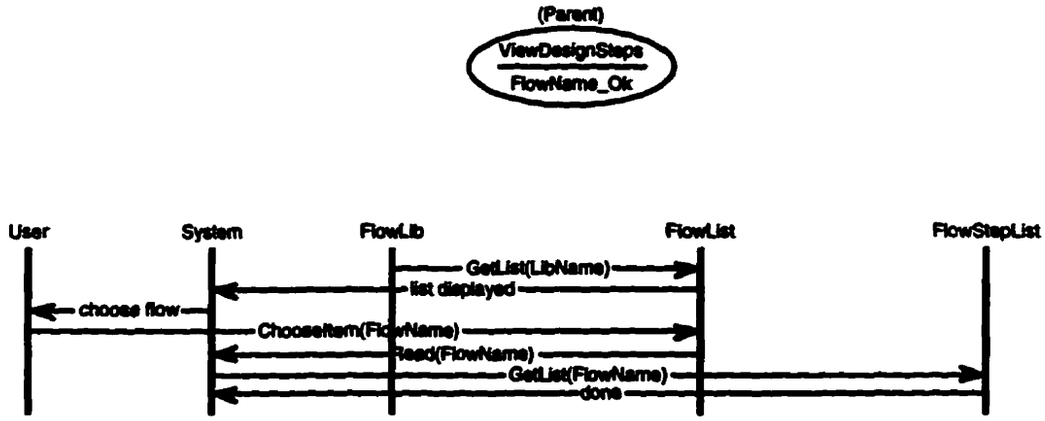


FIGURE 19. ViewDesignSteps use case event trace.

3.2.1.2.14. OpenFlow use case (Figure 20).

The textual description of the actions performed during the OpenFlow use case is:

- the user chooses the OpenFlow menu option,

- the system after successfully performing the ActiveLib use case, requests the FlowLib object to return the list of existing flows,

- the FlowLib object requests the FlowList object to return the list of flows contained in the activated library,

- the system displays the list of flows,

- the system asks the user to choose the flow to be executed,

- the user chooses the flow,

- the system executes the FlowXpert with the chosen flow being activated.

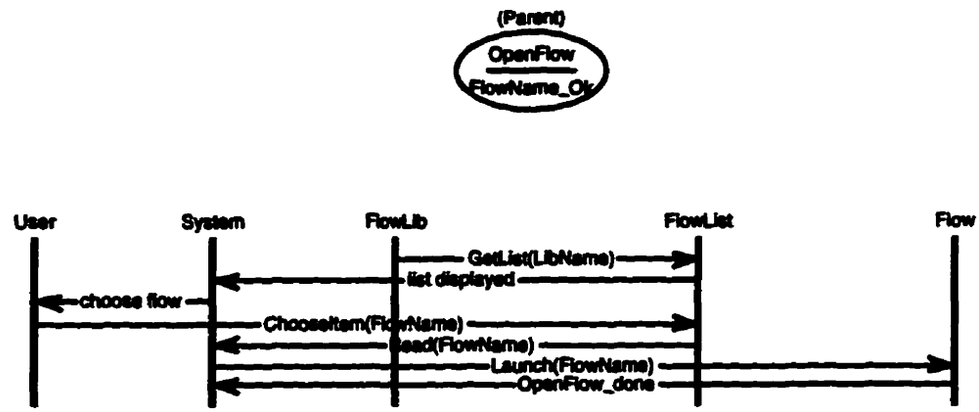


FIGURE 20. Open Flow use case event trace.

3.2.1.2.15. SearchFlow use case (Figure 21).

The textual description of the actions performed during the SearchFlow use case is:

- the user chooses the SearchFlow menu option,

- the system after successfully performing the ActiveLib use case, requests the FlowLib object to return the list of existing flows,

- the FlowLib object requests the FlowList object to return the list of flows contained in the activated library,

- the system creates SearchStepList, SearchMachine and SearchResult objects,

- the system asks the user to choose from the LibStepList the design steps required by his design process and sends them to the SearchStepList object,

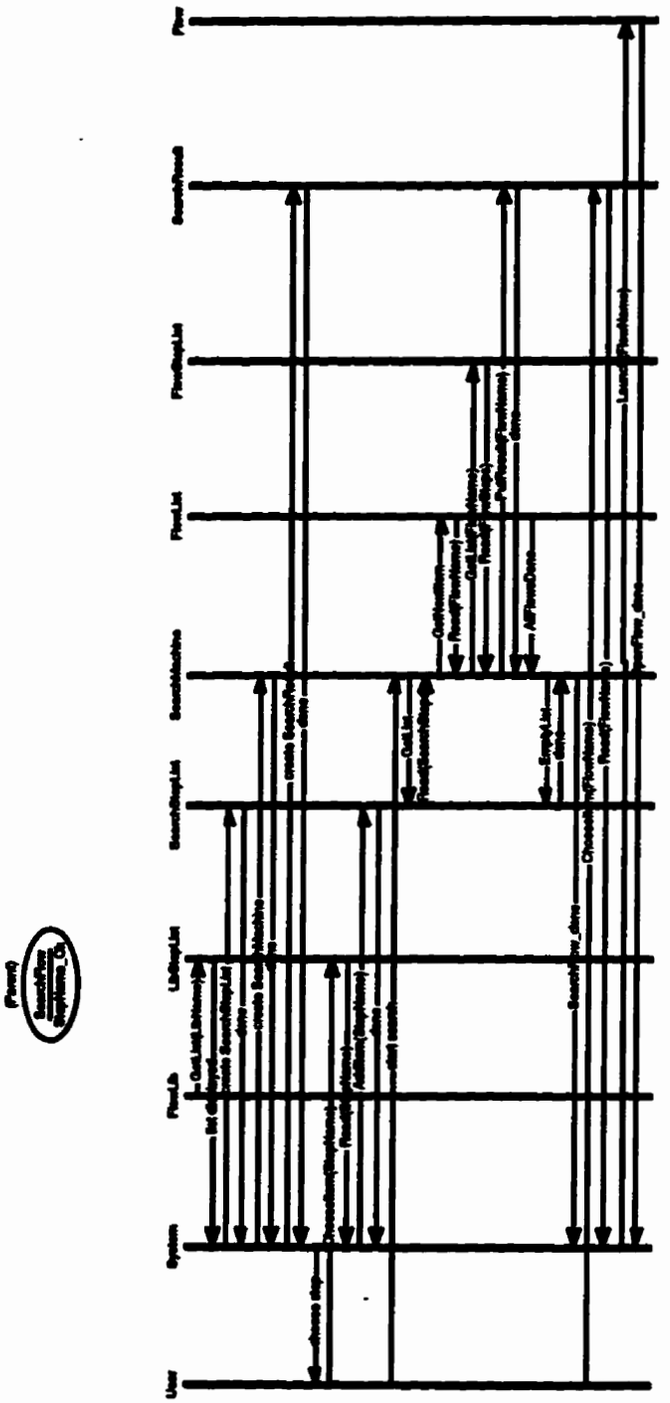


FIGURE 21. Search Flow use case event trace.

- after all steps have been chosen, the user starts the searching procedure which causes the sending of a Start Search event to the SearchMachine object,

- the SearchMachine object reads the steps from the SearchStepList object,

- the SearchMachine sequentially reads flow names from the active library flows contained in the FlowList object,

- after receiving a flow name, the system sends a request to the FlowStepList to send a list of design steps of the particular flow,

- the SearchMachine object compares the required steps to those included in the actual flow,

- the SearchMachine sends the comparison result to the SearchResult object,

- the SearchResult object displays the information that shows the extent to which particular flows meet the requirements of a particular design process in terms of its design steps,

- after extracting all the flows included in the FlowList, the SearchMachine clears the SearchStepList object and sends a message to the system indicating the completion of the search activity,

- the user may choose any flow displayed by the SearchResult object which will return the flow name to the system to enable executing the flow.

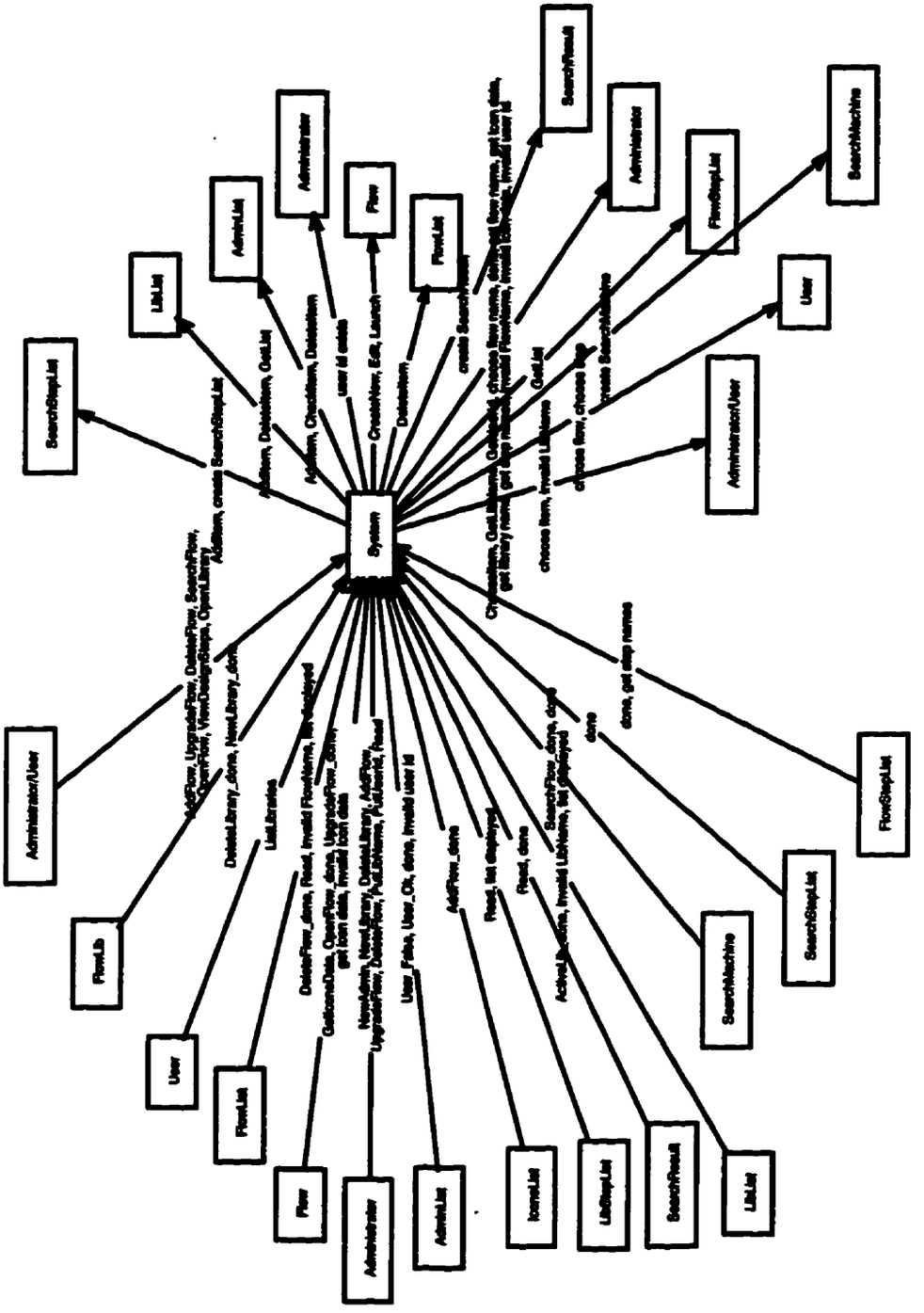


FIGURE 22. The Event Flow - summary of all events taking place for a System object.

3.2.1.3. Graphical user interface.

A preliminary design of the graphical user interface (GUI) is very important at this point. Visual representation of the system "like the user will see it", is very helpful in interacting with the potential users to extract and complete their expectations and requirements [7]. Delivering to the users the GUI without functionality behind it but with all the feel of the system is very encouraging and helpful for them to generate feedback and possibly new requirements.

At the preliminary stage the GUI builds upon use cases, corresponding to menu options, which capture non obvious interaction with the user. The main window, search flow, add flow and view design steps are considered as such. The main window (Figure 23) contains four different sections. The top space is occupied by the menu bar providing a means to activate all the menu options. The central part of the window is occupied by two sections - the left one displaying flow icons from the activated library and the right one being a simplified file manager. The flow icons enable the activation of flows by employing a pointing device. The bottom of the window consists of a status message box which displays information regarding actual tool status and activities, as well as execution errors.

The search flow menu option GUI (Figure 24) contains two listing boxes, one displaying all the design steps contained by all the flows included in the activated library, the second displaying user choices of required design steps necessary to accomplish his design process. The bottom of the window consists of a Search Result text box which displays statistical results of the search flow activity. The three buttons enable starting search activity, canceling the use case and returning to the main window.

The add flow menu option GUI (Figure 25) contains three entry fields to acquire from the user all the necessary information related to the new flow such as: flow name, flow location and the flow's icon location. The radial buttons enable the setting of the number of design steps contained in the new flow. The window ptompting for the names of design steps is activated by the "Enter flow step names" button.

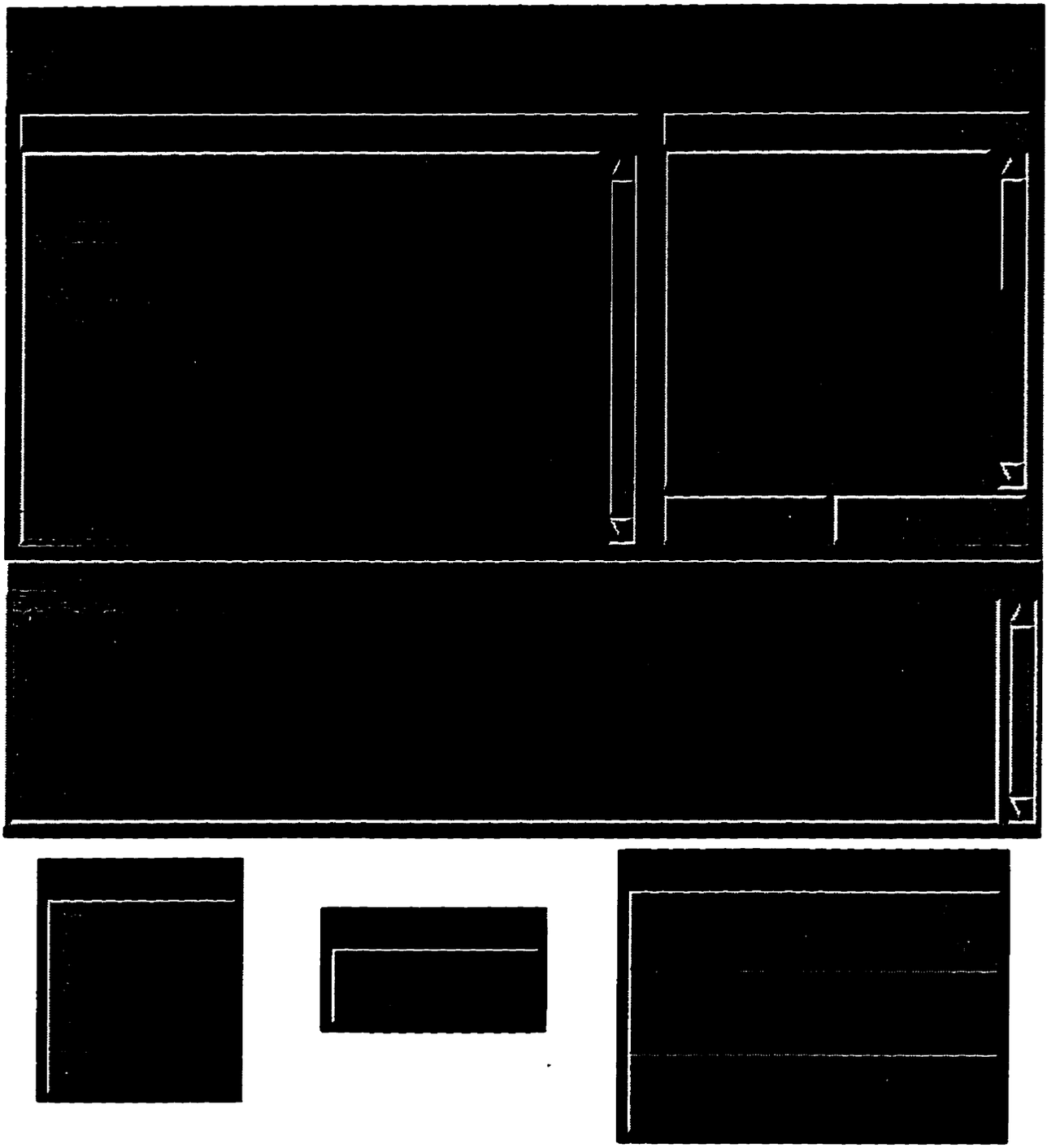


FIGURE 23. The GUI of the main window along with the menu options.

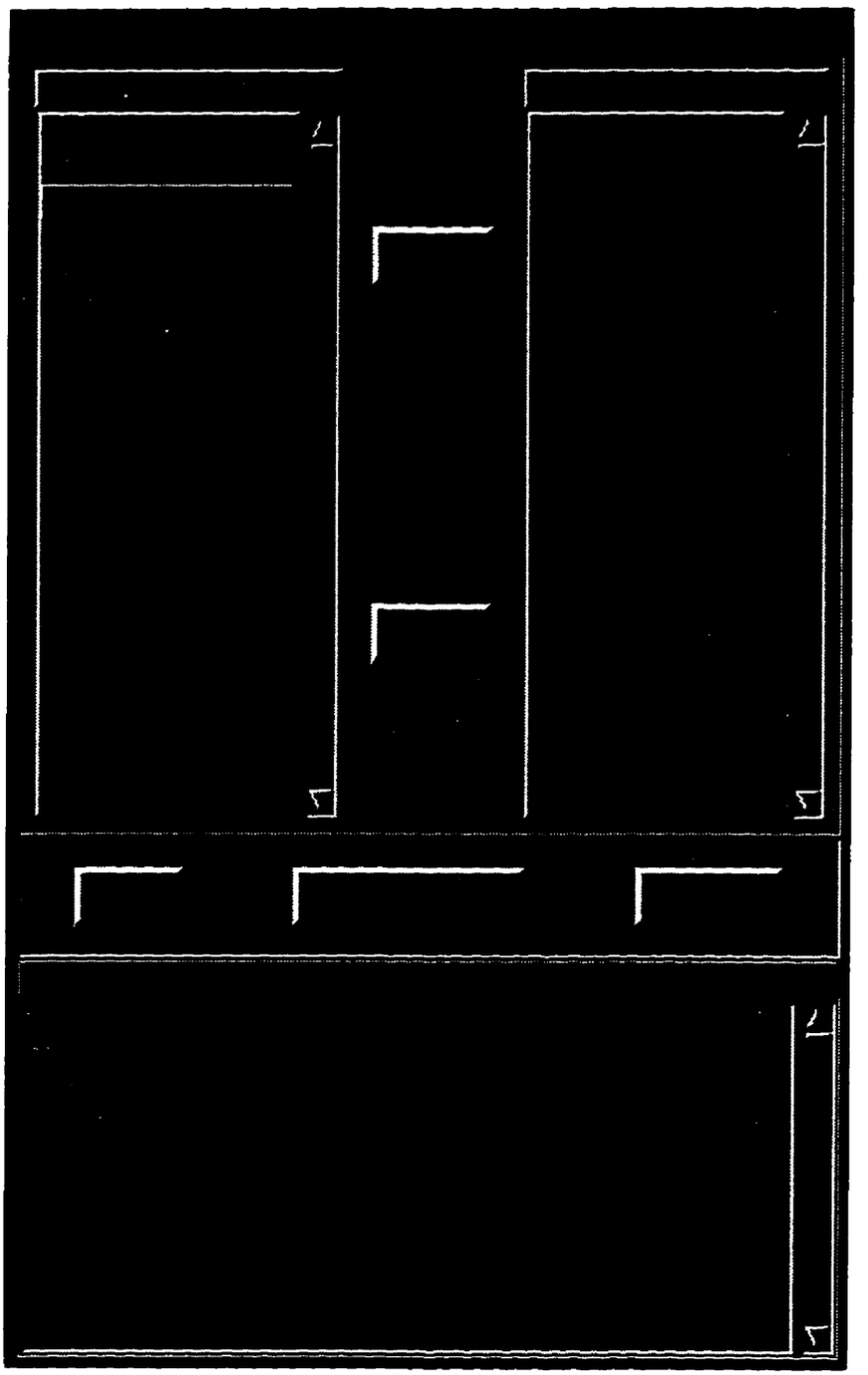


FIGURE 24. The GUI of the Flow Search menu option.

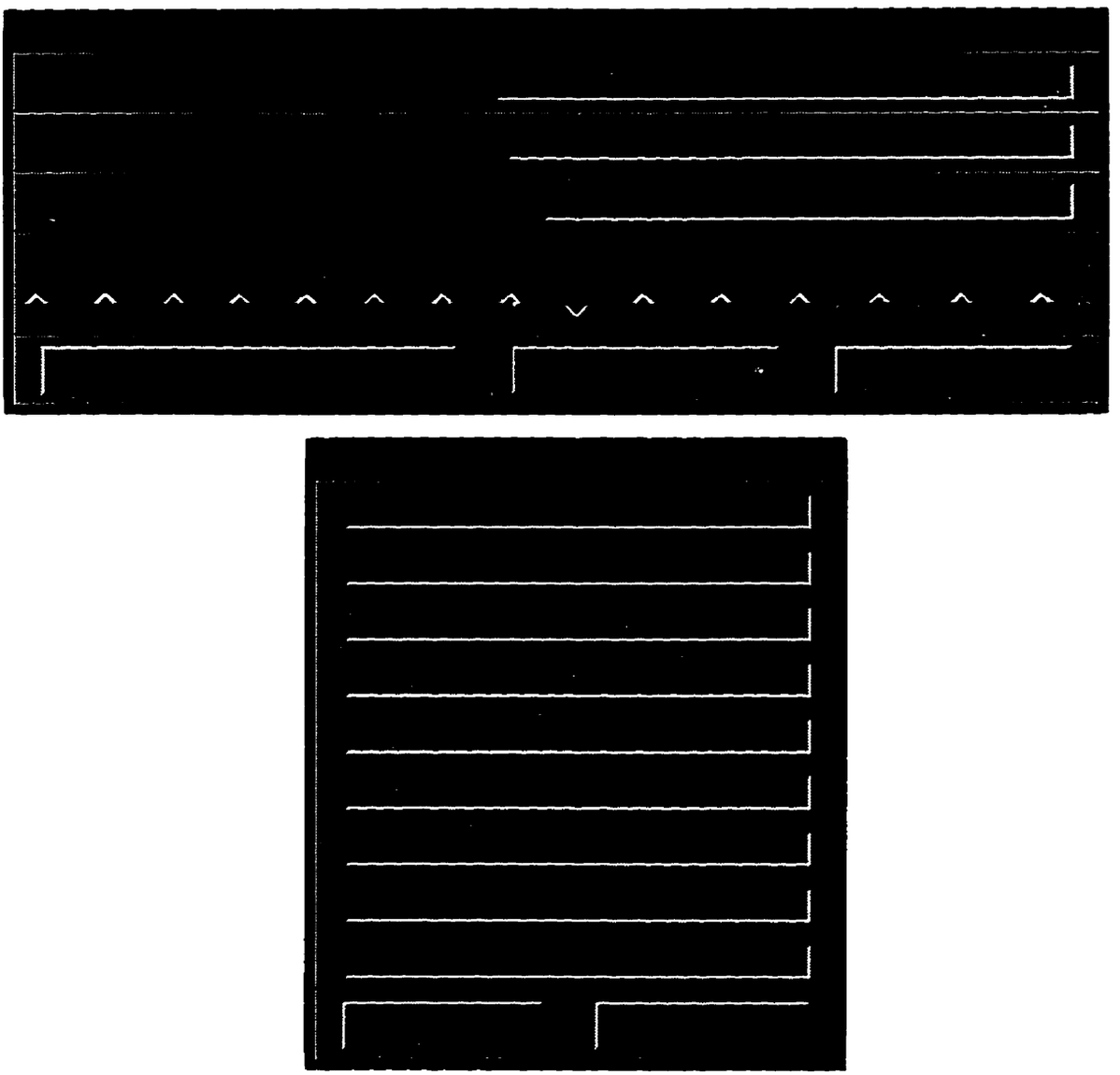


FIGURE 25. The GUI of the Add Flow menu option.

The view design steps menu option GUI (Figure 26) consists of two list boxes which contain flow names that belong to the activated library and step names of the chosen flow.

3.2.2 System analysis.

The System Analysis objective is to build a model of the system which expresses its functionality and internal structure but which is not affected by implementation issues. Designers using the OMT notation analyze the system by creating three separate models which are the Object, Dynamic and Functional Models [8]. In this phase the first version of these models is created. Together they form an Analysis Document which is the deliverable of this step [6].

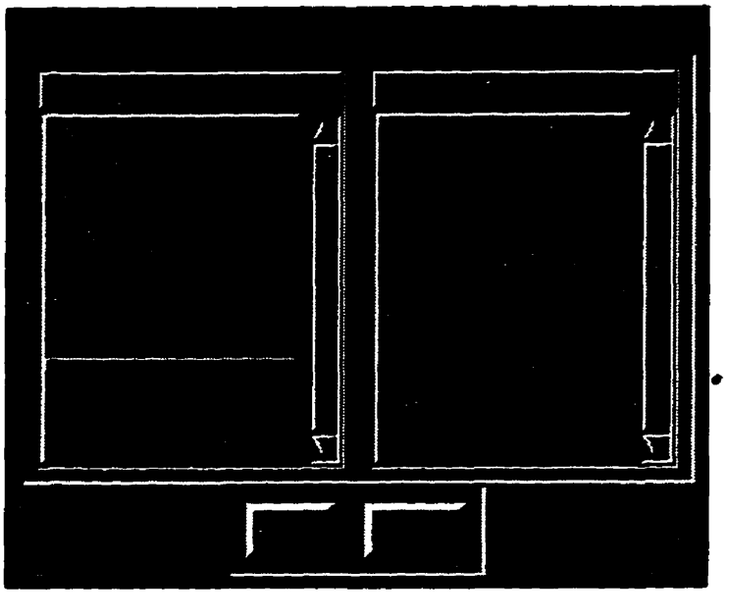


FIGURE 26. The GUI of the View Design Steps menu option.

3.2.2.1. Object model.

The Object Model defines WHAT the basic components of the system are, and the kind of associations that exist between them. The Object Model represents the static system structure [6] [8]. The components called Objects, appear in the form of structures consisting of member data and member functions operating on these data [8][11]. Objects are extracted directly from the Requirements Statement and Use Case Scenarios. They appear in the Use Case Scenarios as interacting entities.

In many cases those entities correspond to the system's persistent data objects [7]. A very efficient way to extract objects is to consider all the nouns appearing in the Requirements and Use Cases as potential objects. In the set of the nouns, objects should be distinguished from object attributes and irrelevant objects from the application point of view. Defining objects helps to find and define associations between them. Associations represent static relations between objects. Objects sharing similar characteristic in terms of member data and member functions may be grouped into Object Classes from which particular objects are instantiated thereby significantly improving code reuse [8][11].

The Object Model built based on the use case analysis, illustrates the classes of objects existing in the use cases as active actors and the static relationships existing between them (Figure 27). Three different kinds of associations exist in the model: one-to-one, one-to-many and composite association. The FlowLib is composed of LibStepList, FlowList and IconList. The LibList is a class of storage objects which preserve lists of objects of the class FlowLib. The FlowLib is a class of composite objects which represent flow libraries and consist of three components: LibStepList, FlowList and IconList. The LibStepList is a class of storage objects which preserve lists of design steps contained by all the design flows belonging to a particular flow library. The FlowList is a class of storage objects which preserve lists of all flows belonging to a specific flow library. The IconList is a class of storage objects which preserve lists of icon data corresponding to all the flows contained by specific flow libraries. The FlowList contains Flow objects which is a class of objects preserving data specific to flows. The Flow objects keep their icon information in an object of type IconList. The Flow has a FlowStepList which is a class of storage objects preserving lists of design steps contained by the specific design flow. The FlowStepList can be accessed by the SearchMachine which is a class of objects encapsulating functionality necessary to compare sets of required steps to those contained by flows. The set of required design steps is preserved in a storage object of the class SearchStepList. The SearchMachine generates a SearchResult which is a class of objects preserving information containing SearchMachine activity results. Three classes of objects contain lists of design steps. FlowStepList, LibStepList and SearchStepList contain one or more objects of class DesStep. The class DesStep represents objects preserving information

about design steps. The AdminList is a class of storage objects preserving lists of users who have been granted administrative privileges. The User is a class of objects preserving user names. Hypothetical procedures and attributes of the object classes have been extracted from the use cases (Figure 28). The hypothetical procedures represent events defined in the use cases and attributes represent values used as events arguments.

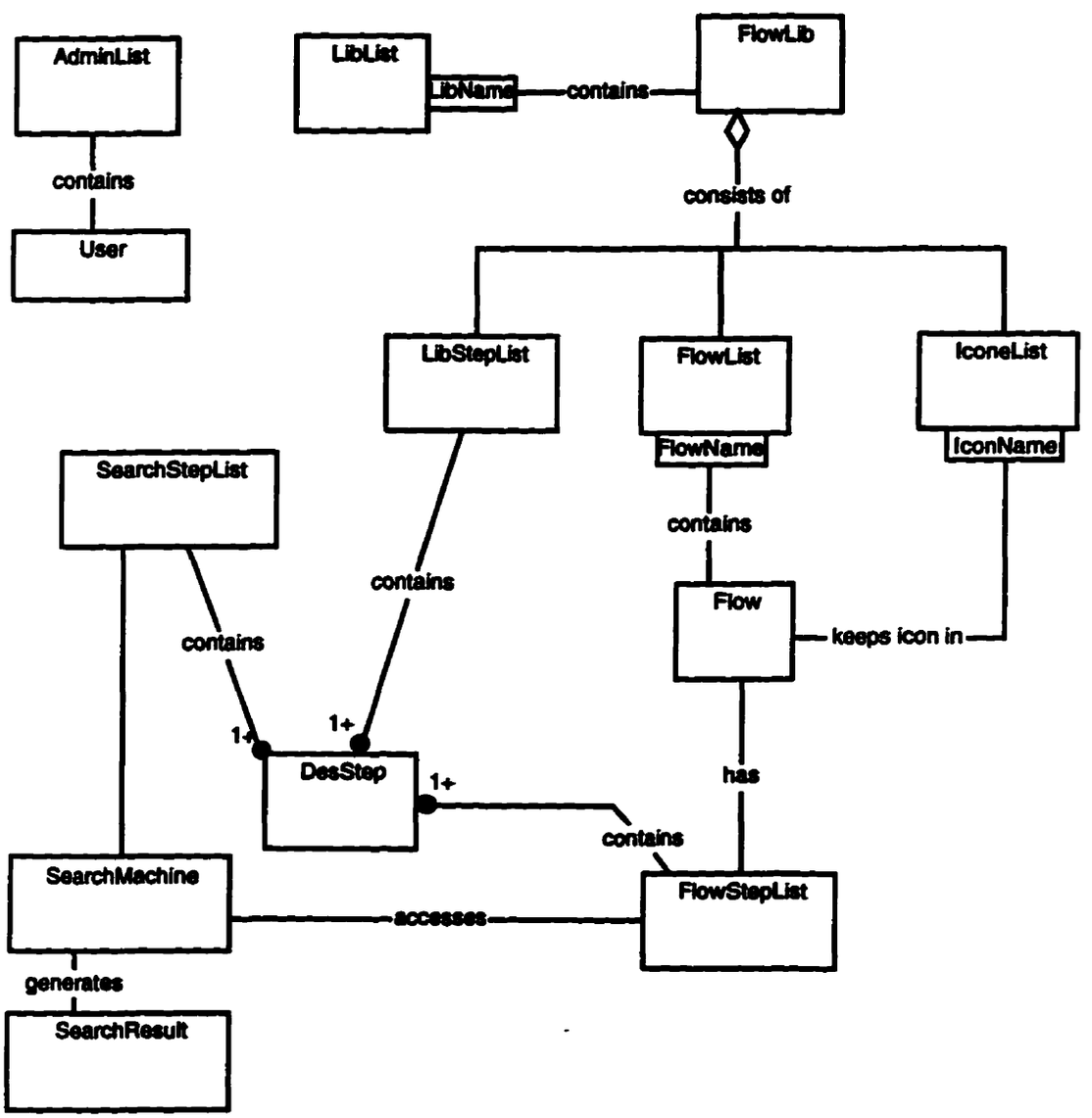


FIGURE 27. The analysis stage Object Model.

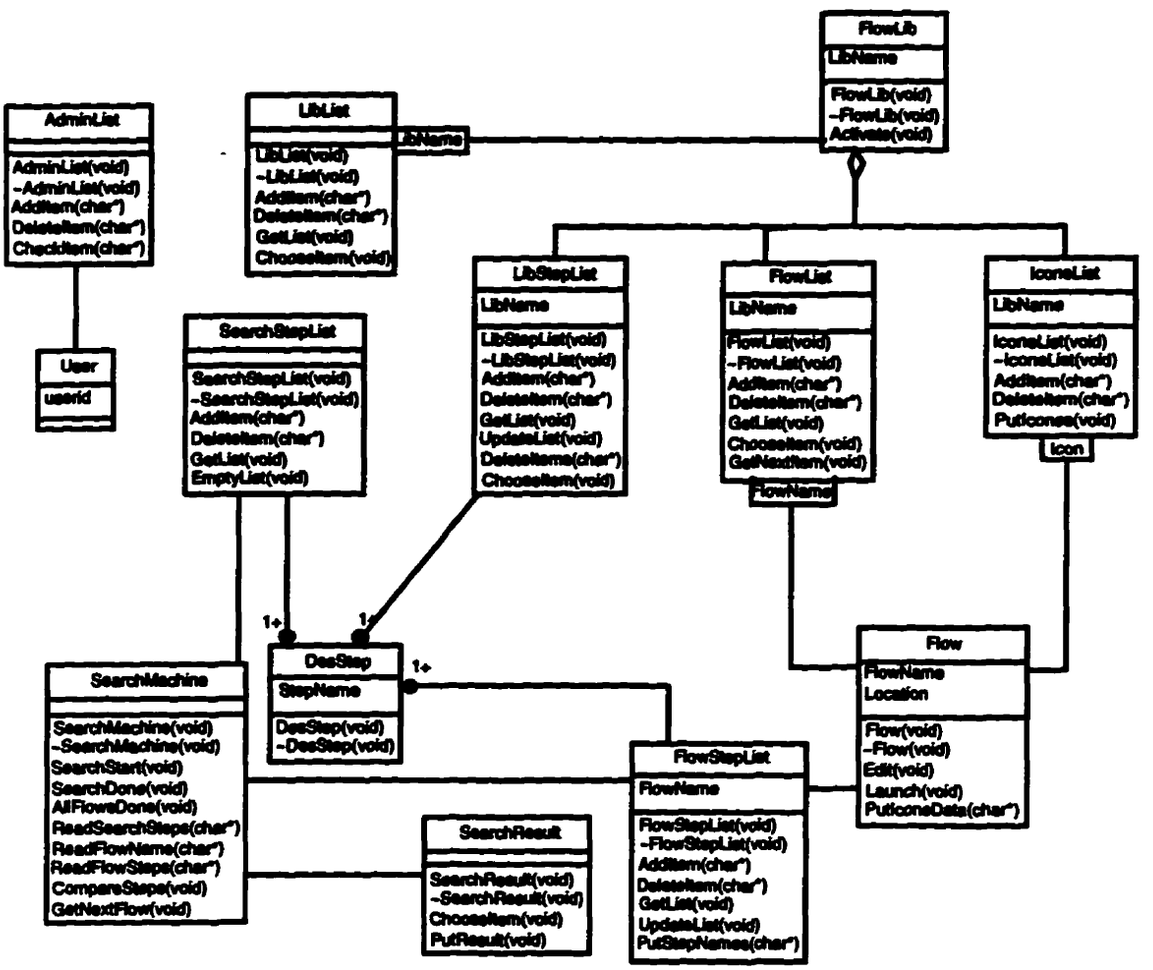


FIGURE 28. The analysis stage Object Model after introducing hypothetical class procedures and attributes.

3.2.2.2. Dynamic model.

The Dynamic Model represents the system's operations which are divided into elementary events that in most cases change the system's internal state. The model shows WHEN those events take place and what kind of object transitions they cause [8]. The dynamic model consists of Object State Diagrams [6] created for all the class objects with behavior which is not obvious. The diagrams represent

objects as Finite State Machines and represent all the events received and sent by objects in all the use case scenarios and captures an object's internal state changes initiated by those events.

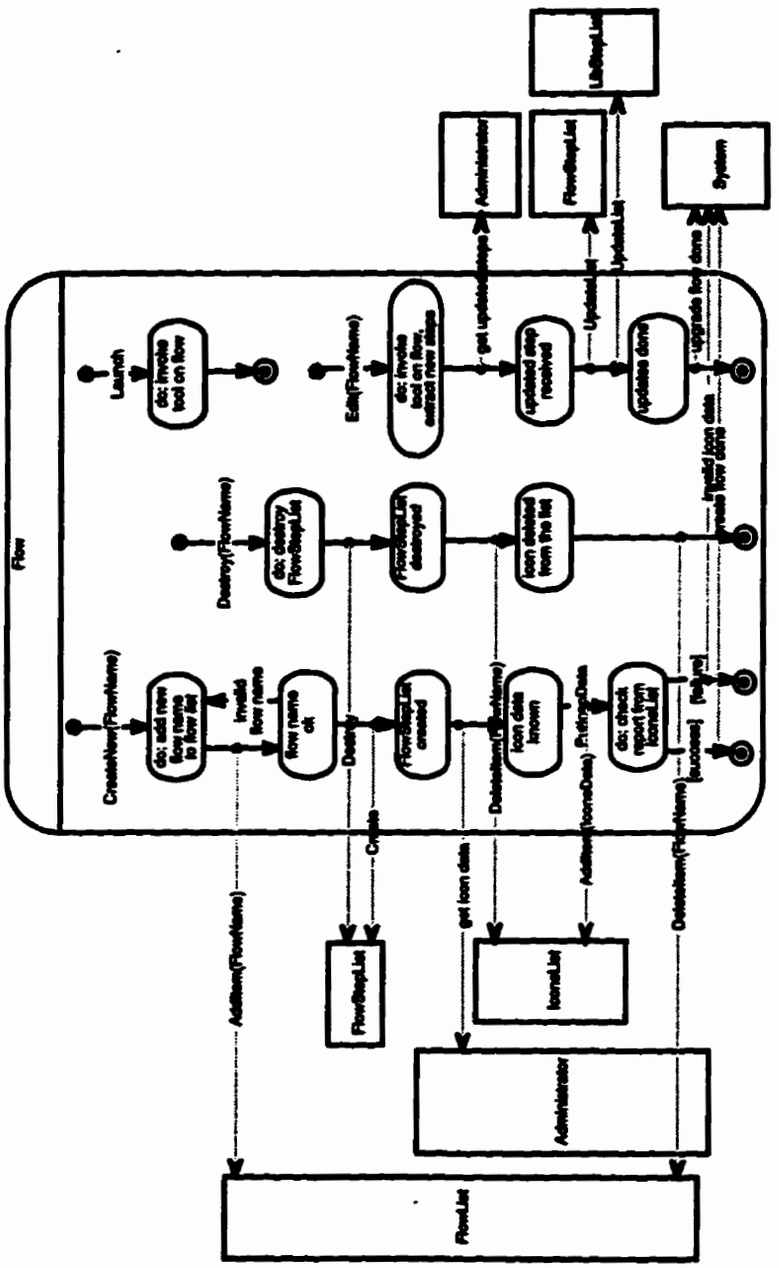


FIGURE 29. Object State Diagram of an instance of class Flow.

Different internal states of an object correspond to different values of its data members [6][8]. In the case of the Design Flow Manager, the dynamic models of Flow and SearchMachine objects have been created.

The state diagram of an object of class Flow (Figure 29) presents 4 different events which initialize activities leading to changes in the flow's internal state. Those activities are: CreateNew, Destroy, Launch and Edit.

A CreateNew event begins a sequence of activities leading to new flow creation. Along with the event signal, the value of FlowName is passed to an object of class Flow. Receiving the event signal pushes the Flow object to the "add new flow name to flow list" state. In this state, the Flow object sends an AddItem event along with the value of FlowName to the object of class FlowList. This event causes the FlowList object to determine if the FlowName value corresponds to any of the items contained in the list. In the case, that the name is not included, a new object is attached to the list and a "flow name ok" message is returned to the Flow object which goes to the next state. Otherwise, an error message is returned and the Flow object returns to the previous state prompting the user again for the value of the FlowName variable. In the "flow name ok" state, the Flow object sends a Create event to the FlowStepList class, which causes the creation of an object of this class. After receiving confirmation of the FlowStepList object creation, the Flow object requests icon data from the User. After receiving icon data, the Flow object sends the data to the IconList object as an argument of an AddItem event. The IconList object checks if the received icon data corresponds to any of the elements already contained in the list and if not it attaches a new element to its list and returns control to the System. In the case that the icon data is already correlated to anyone of the items in the list, an error message is returned to the System.

A Destroy event begins a sequence of activities leading to the elimination of an existing flow from a flow library. The FlowName value is passed to the Flow object as an argument of the Destroy event. Next, the Flow object sends a Destroy event to the FlowStepList object. After successfully destroying this object, the Flow object asks the IconList object to delete from its list the icon data corresponding to the

Flow object. Then the Flow object sends an event to the FlowList object which causes the deletion of the item corresponding to the flow from the list.

A Launch event begins a sequence of activities leading to an existing design flow execution. In the state initialized by this event, the Flow object passes its name to the FlowXpert for execution.

An Edit event begins a sequence of activities leading to editing an existing design flow for changes and updates. Along with the event, a FlowName value is passed as an argument. The Flow object launches the WorkXpert tool, which edits the flow. After the introduction of the flow changes and termination of WorkXpert, the Flow object requests the Administrator for information regarding changes in terms of design steps. After receiving information from the Administrator, the Flow object sends events to the FlowStepList and LibStepList objects requesting their lists to be updated. Successful updates make the Flow object returns control to the System.

The state diagram of an object of class SearchMachine (Figure 30) represents changes in the internal state of the object. A Search event initializes the SearchMachine constructor. After successful construction of the SearchMachine object, the object reads the active flow library name which is the value of the FlowLib variable. Next the SearchMachine object requests the list of required steps from the SearchStepList object. After receiving the step list, the SearchMachine object requests the FlowList object for the next item which is a flow name contained in the active library. Upon receiving the flow name, the SearchMachine object requests the FlowStepList object for the list of steps building the flow. Next, the SearchMachine object compares both lists of design steps and after generating a result, sends it to the SearchResult object to be displayed to the User. At this point, the SearchMachine object goes back to request the next flow name from the FlowList object. The process is repeated as long the FlowList object provides the SearchMachine object with flow names. Once all the flow names have been processed, the SearchMachine sends a completion message to the user and clears the content of the SearchStepList object to make it ready for the next search case.

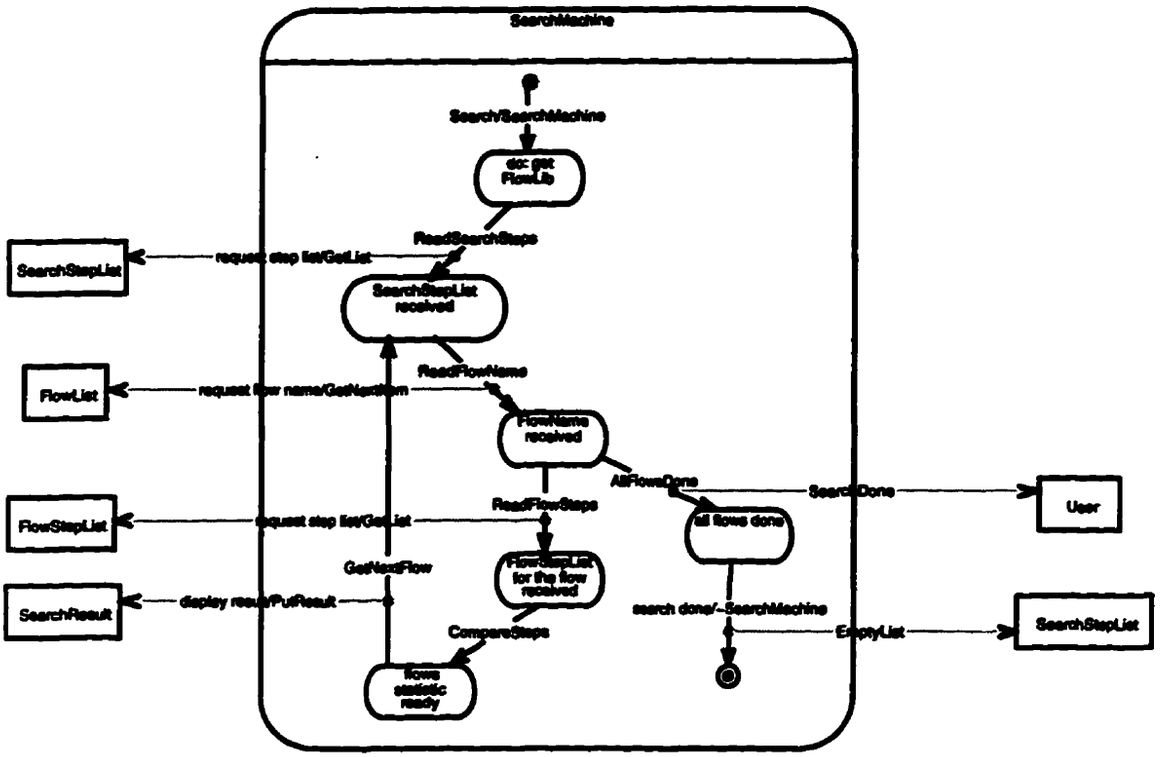


FIGURE 30. Object State Diagram of an instance of class SearchMachine.

3.2.2.3. Functional model.

The Functional Model shows HOW the system operations are performed. To achieve this, Data Flow Diagrams (DFDs) are created for system operations and are particularly useful for those that are more complex. Data Flow Diagrams are the functional decomposition of the system and represent the kind of data transformations are performed during the system operation [6][8].

For some systems it is not necessary to create a Functional Model. This is the case for systems with functionality focused on data management as opposed to data transformation. In the case of systems where interaction with the user is very

advanced and constitutes the main part of the system functionality, the Functional Model is very helpful [7].

As an addition to the Data Flow Diagrams, Operation Descriptions may be created. These are textual descriptions indicating what the operations are supposed to achieve, their inputs, outputs and which objects are modified by them, as well as the conditions necessary to perform them [6].

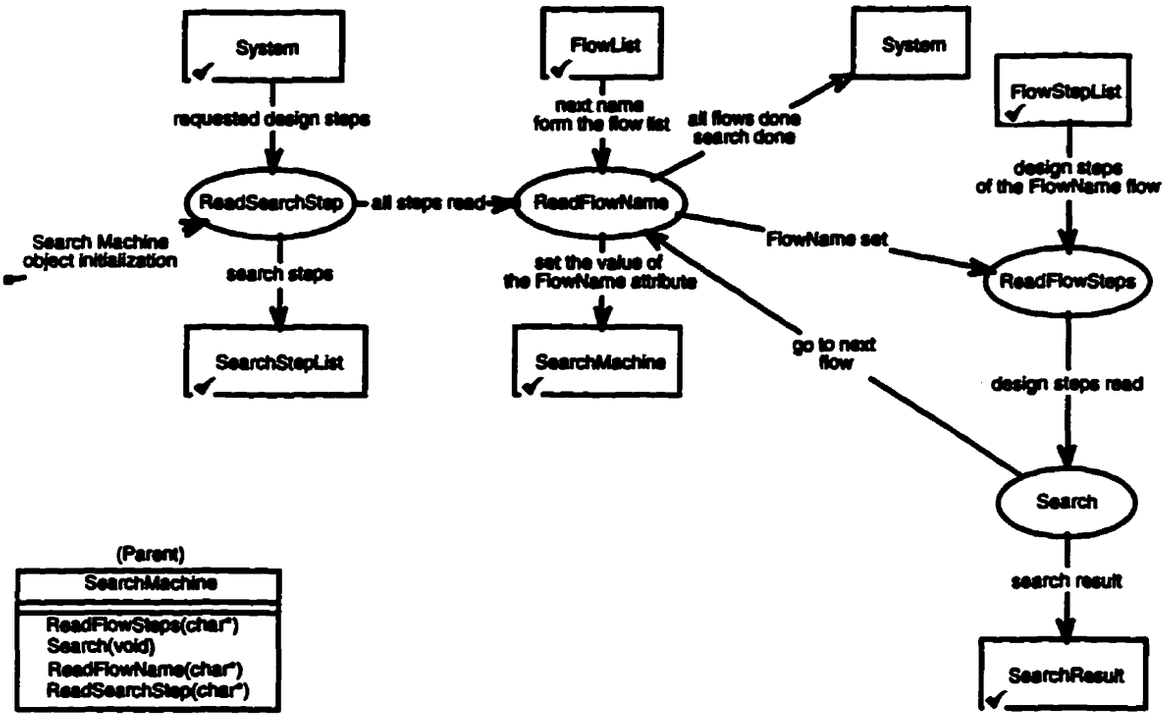


FIGURE 31. Functional Model of the Search Flow operation.

The Design Flow Manager was not considered to have functionality complex enough to make creating a functional model for all the operations necessary. The only operation complex enough is Search Flow (Figure 31). An object of class SearchMachine is initialized as the first step of the Search Flow operation. Next, the ReadSearchStep procedure receives from the System requested steps one by

one and sends them to the SearchStepList object to be attached to the list. Upon reading all the steps, the ReadFlowName procedure receives a Flow object from the FlowList object and sets the FlowName attribute of the SearchMachine object to the value of the Flow object name. Next, the ReadFlowSteps procedure reads design steps from the FlowStepList object which is a composite of the Flow object. After all the steps have been read, the Search procedure performs the comparison of step sets and sends the result to the SearchResult object to be displayed. Next, the control goes back to the ReadFlowName procedure to extract the next flow name from the FlowList object. Once all the flow names have been read from the FlowList object, the operation terminates returning control to the system.

3.3 Design.

FIGURE 32. Design Phase.

3.3.1 System design.

During the System Design phase, the main system strategic decisions are to be considered. A System Design Document is the deliverable created during this step [6].

3.3.1.1. System Decomposition.

In many cases the internal structure of the system is so complex that system decomposition may be very helpful in a further design process. Since the complexity of the system is already known, the designing resources may be allocated. Subsystems extracted from the original system should be consistent in terms of their functional-

ity. A good indication of a well done decomposition is a minimal number of event interactions between different subsystems.

At this point in the design it is the perfect time to improve some of the software qualities, such as maintainability, repairability, evolvability and reusability. This may be achieved by defining interfaces for all the subsystems and configuring them in the Client - Server mode [21]. Subsystem interfaces strictly define all the services provided by subsystems and what kind of information/data has to be provided with each service request. In the Client - Server approach, the client initializes communication with the server whenever it needs to use a server's services. In this configuration all the changes introduced to the system are done locally which has a significant impact on extending the above mentioned qualities.

The Design Flow Manager in its first release is designated to be used in the university environment. The concept will be evaluated by applying the tool to organize CAD expertise access to students both taking CAD courses or laboratories and graduate students performing their own designs.

These assumptions led to the idea of separating the functional algorithm and representation to enable higher maintenability, changeability and incrementality. To decrease the risk factor and apply reusability, the scheme for the tool architecture has been taken from a design patterns library. Design patterns and especially architectural design patterns were introduced which provided designers with a continuously increasing number of patterns that address the most common aspects and problems of software architecture. The Model/View/Controller (MVC) [15][16] used by Smalltalk-80 to build graphic user interfaces is a perfect example of a pattern which addresses the Design Flow Manager requirements (Figure 33). The system is partitioned into three main subsystems. The Model represents the application in terms of the system's functionality. The View defines the representation of the system in the form of a graphic user interface which is independent from Model. The Controller is responsible for interpretation of user interactions with the system. Any of the subsystems may be independently changed both statically and dynamically. The independence of the subsystems also enables the employment of different technologies during their implementation. This advantage will also be used in

the design of the tool. Since the best feedback from the user is acquired in the form of evaluation of the interaction with the software through the graphic user interface, the implementation of the GUI should enable fast and easy responses to user requests. The tcl/tk [17] scripting language which provides a platform for rapid GUI prototyping, meets all the afore mentioned requirements. Since the View is designed for flexibility, the Controller should be designed for flexibility too.

The Model has been implemented in C++ to improve system performance. To preserve system changeability in terms of its functionality, the Model subsystem is implemented as a set of compiled modules which may be updated according to changes in the set of menu options.

The three subsystems interact with each other and the user communicates with the system through the View. The implementation languages of the subsystems are specified as follows: View is implemented in Tcl/Tk, Model in C++ and Controller is implemented in both languages.

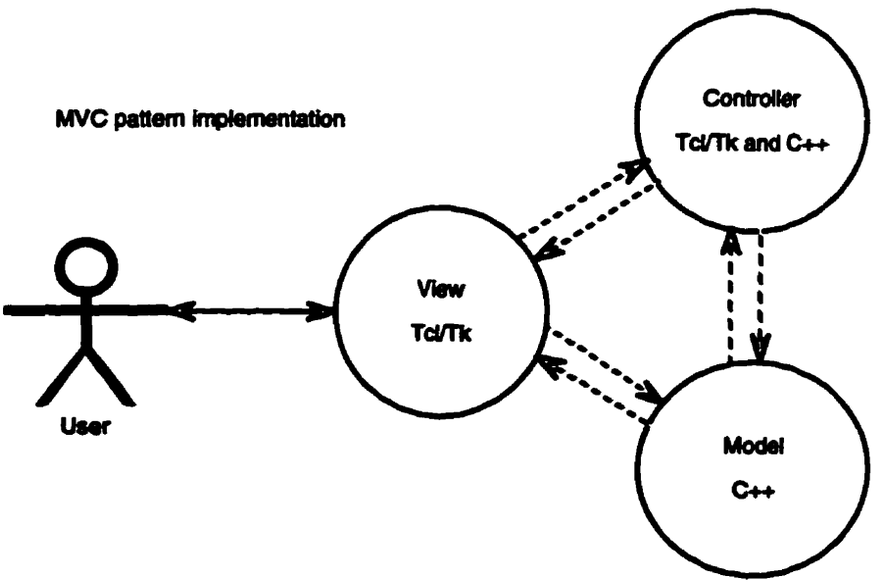


FIGURE 33. Software architecture of Design Flow Manager.

3.3.1.2. Specifying Concurrency

There are many ways that concurrency may occur in the system. Use cases may have concurrency existing within them as well as different use cases or different instances of the same use case may be active at the same time. This leads to the situation where different objects belonging to the same or different classes and even whole subsystems may be concurrent. The Object Model is a good place to trace concurrent objects.

Good management of the concurrent entities may be very critical for real-time systems. For those, in some cases the concurrent subsystems may require separate hardware to be able to meet the time constraint requirements.

Concurrency effects are not expected to occur in the case of Design Flow Manager.

3.3.1.3. Task - Resource Allocation

In the case of complex systems or systems whose requirements expect distributed resources, Task - Resource Allocation has to be performed. In the simplest case the tasks correspond to subsystems. The allocation may take place between distant networked computational units or different processors coexisting in the same unit.

The Design Flow Manager is designed with the purpose to be run locally on a single station although it provides the means to access design flows distributed across a local network.

3.3.1.4. Data Stores Implementation Strategy

Persistent data storage also has to be considered. The choice between files, relational database or object-oriented database has to be made. For applications using small amounts of data, files are recommended because of their simplicity. In other cases there is a choice between relational and object-oriented databases. The relational database information representation is table oriented where tables may contain primitive data types. There is no easy way to store objects with all their attributes in the table structures. This incompatibility creates the need for an inter-

face which will enable transitions between object oriented and table oriented data representations [7].

For the purpose of the Design Flow Manager, a file system has been chosen to store persistent data. The choice was motivated by the relatively small amount of data to store. The file system consists of files and folders dedicated to libraries and flows (Figure 34). Each flow library posses a dedicated folder which contains the flow_list.names file and folders dedicated to each flow belonging to the library. The flow_list.names file preserves information about which flows belong to the library. Each flow folder contains three files: flow_step_list.names, icon.dfl, location.dfl. The flow_step_list.names preserves information about all the design steps included in the flow. The icon.dfl and location.dfl preserve information about the flow's icon and executable file location respectively.

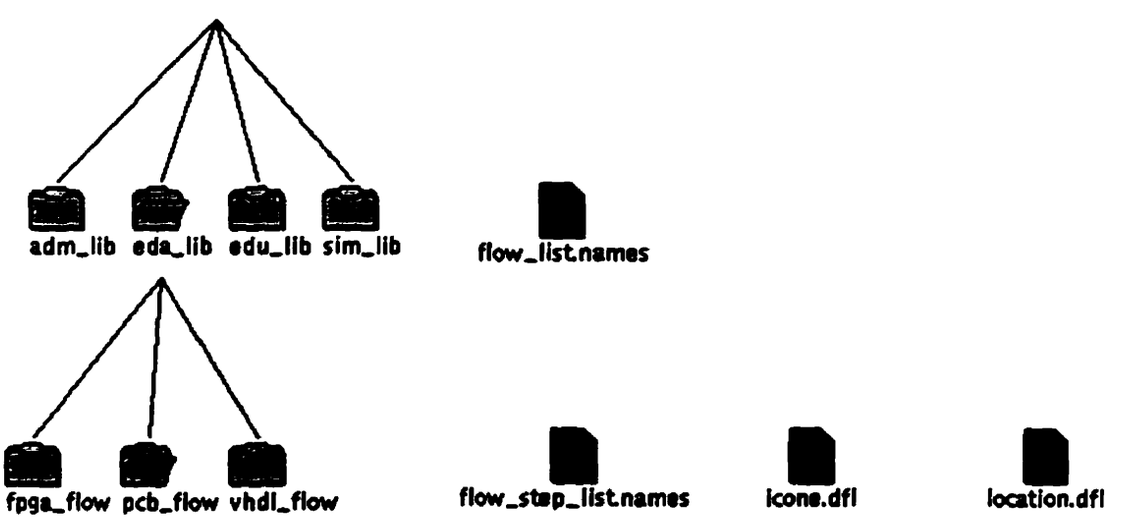


FIGURE 34. Library folders and file, flow folders and files - persistent data storage file model.

3.3.1.5. Software Control Approach

The choice of software control has to be made. In many cases event driven control is applied especially for systems with developed graphical user interface. The GUI may be designed as the global control object which activates specific procedures in the response to inputs received from the user and environment. Real-time systems, especially those characterized by concurrency may need a concurrent control approach to be applied.

The architecture of the Design Flow Manager separates control from view and model. The event driven controller defines the interface between the GUI implemented in tcl/tk and compiled C++ programs representing implementations of the menu option actions of the tool. The interfacing is implemented partially in tcl/tk and C++ where the tcl/tk event loop is used to implement system events tracking.

3.3.1.6. System Behavior in Exceptional Situations.

System behavior in situations like start up and abnormal termination as well as error handling has to be determined. The procedures to release resources should be defined. Memory deallocation and external resource link elimination belong to the most important cases.

During start up the Design Flow Manager needs to be able to read the DFL_HOME and USER environment variables. The DFL_HOME variable defines the location of the software in the system and should have its value set prior to tool initialization. The USER variable should contain the actual user id which is necessary to test user administrative privileges. The activated library is set initially to eda_lib which contains the PCB design and analysis flow described in chapter 2.

In the case of any error during tool execution, the corresponding message is displayed to the user either in the status text box at the bottom of the main window or dialog boxes generated by the tcl/tk interpreter. Failure to provide necessary entry data will prevent the tool from executing a particular program action.

At this stage, no mechanism is defined that assures memory deallocation after system termination except for standard initialization of object destructors taking place before program termination.

3.3.1.7. Other Strategic Decisions

Other strategic decisions reflect considerations concerning operating system platform, system portability, system performance and resource requirements.

The Design Flow Manager was implemented and designed to be used on Sun Sparc stations running SunOS or Solaris operating system. The software may be ported to other operating systems supporting the tcl/tk scripting languages and Mentor Graphics flow tools (WorkXpert and FlowXpert). Certainly the system may also be redesigned to support the management of flows designed with the help of software different from the afore mentioned. Compiling the C++ programs on different platforms requires the Rouge Wave foundation class library to be available [18].

Since the software provides only a means to access design flows, the performance has not been considered as a design issue.

Installation of the software requires approximately 3 MB of free hard disk space. There must also be some memory reserved for the persistent data file structure.

3.3.2 Object design.

In the Object Design phase the main focus is put on refining the system model to introduce implementation issues. Also the implementation aspects of Objects are considered which mainly focus around their member functions and attributes. Members' definitions and partitions between Private and Public are specified.

The Object Model is refined and Object Inheritance and Aggregation is introduced to optimize the code reusability factor.

The Object Design phase may require a few refinement cycles to coordinate member function definitions and class generalization.

3.3.2.1. Class associations definition.

In addition to the associations which reflect functionality and structure of the system, inheritance and aggregation associations may be introduced at this point. Both of them radically improve code reuse quality of the software. Objects building up the system may be grouped into Object Classes. An Object Class possesses all the common member data and member functions existing in all the objects instantiated from the class. From an object class other object classes may be inherited. Those new subclasses inherit all the data and function members of the superclass and may additionally have their own members specifically defined for them. A Superclass is a generalization of its subclasses and sometimes the inheritance is called "a kind of" association. The inheritance may be built with many levels ending up with classes with very significant differences in their members. From classes on each level of inheritance, objects may be instantiated by defining values of object attributes. Sometimes, to simplify an inheritance tree Abstract Classes are created, from which no objects will be instantiated [8][9][11][12][19][20].

Aggregation is called "a part of" association and enables the object oriented representation of the internal structure of complex objects. Aggregation associations connect complex objects with objects which are their building components. Aggregation may also be built on many levels.

The study of the object model created during the system analysis stage (Figure 28), leads to the conclusion that the system contains 7 classes of type list. The main task of these lists is to store objects. At this point reusability by inheritance may be introduced. All of the list classes may be inherited from one superclass called LinkedList which includes all the procedures supported initially by all the list classes (Figure 35). To decrease risk and radically shorten the design time, an existing class template was used as the superclass for the linked list classes. The RWTValDlist<T> from the Rogue Wave foundation class library is a template of a doubly linked list which meets the necessary requirements [18]. The hypothetical procedures of LinkedList have been replaced by the procedures being members of the RWTValDlist<T> class (Figure 36).

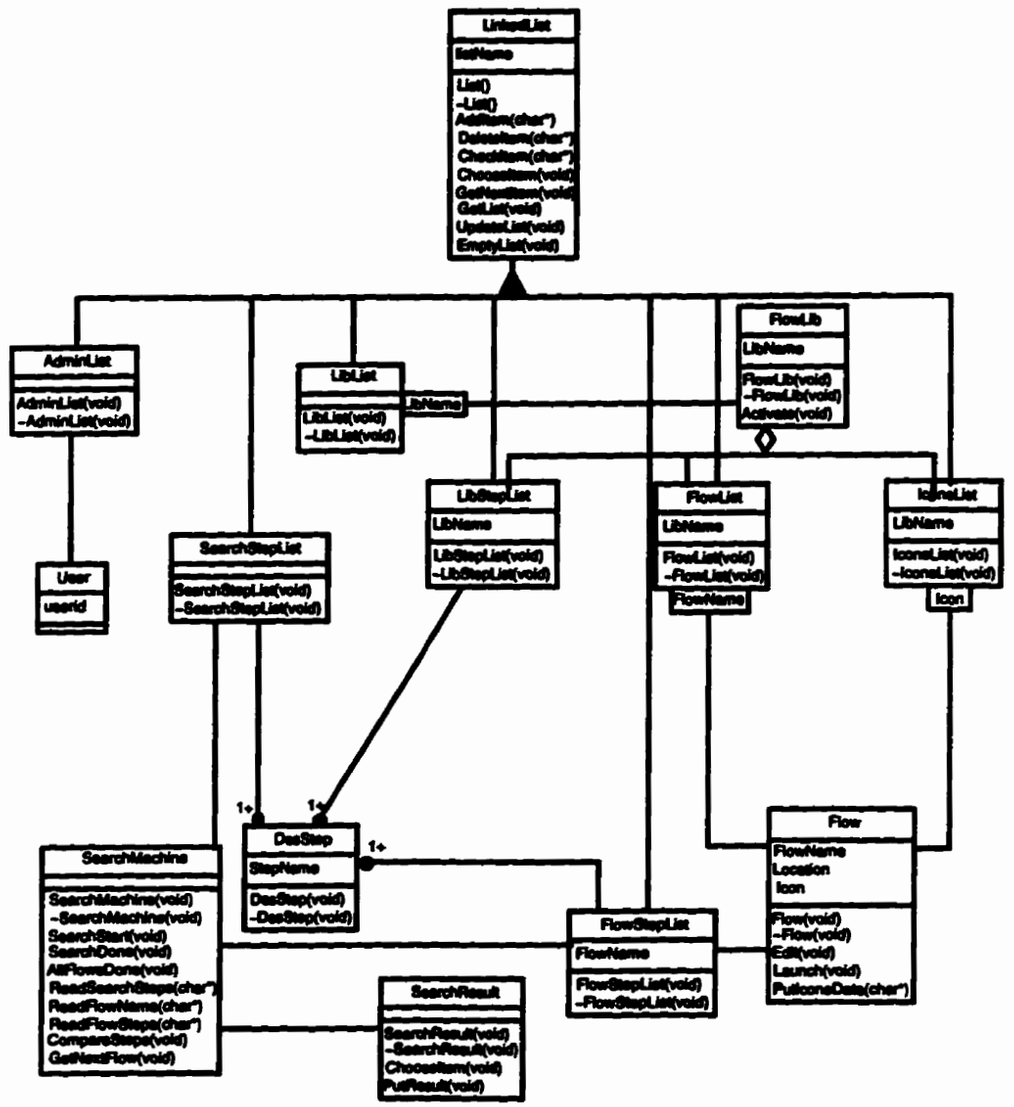


FIGURE 35. Object Model of the model subsystem of Design Flow Manager after introducing inheritance hierarchy for classes of type linked list.

3.3.2.2. Class operations definition.

The events are extracted from the Use Case Event Traces and Object State Diagrams of the Dynamic Model and considered to be function calls. Events received by a particular object are calls to its member functions. Member functions are allocated to objects. Functions are defined as Public Members of the objects. These functions may only operate on the data members of the same object supporting the concept of encapsulation and data hiding. In some cases it may happen that functions should operate on objects belonging to different classes. These functions need to be able to access data members of the objects they operate on. The way to achieve this is to define these as public Friend functions [11][19][20].

The same functions for inherited object classes may require different methods for different classes. In this case these functions have to be defined in the superclasses as Virtual.

The introduction of the RWTValDlist<T> class as the superclass of the linked list classes led to the final revision of their procedures. Procedure arguments have also been defined (Figure 36). The presented object model does not include argument constructors, overloaded insertion operators << and overloaded extraction operators >>. Argument constructors set the object attributes values during the object instantiation process. The insertion and extraction operators overloading procedures are defined as friend functions and enable operations on objects during persistent data files access.[11]

The implementation requirements for the T objects being stored in the RWTValDlist<T> specify that class T must have [18]:

- a default constructor,

- a defined copy semantic (for example Flow::Flow(const Flow&)) which provides a copy mechanism for objects of types defined by the designer,

- a defined assignment semantic (for example Flow::operator=(const Flow&)) which provides overloading of the assignment operator to be used with objects of types defined by the designer,

- a defined equality semantic (for example Flow::operator==(const Flow&)) which provides overloading of the equality operator to be used with objects of types defined by the designer.

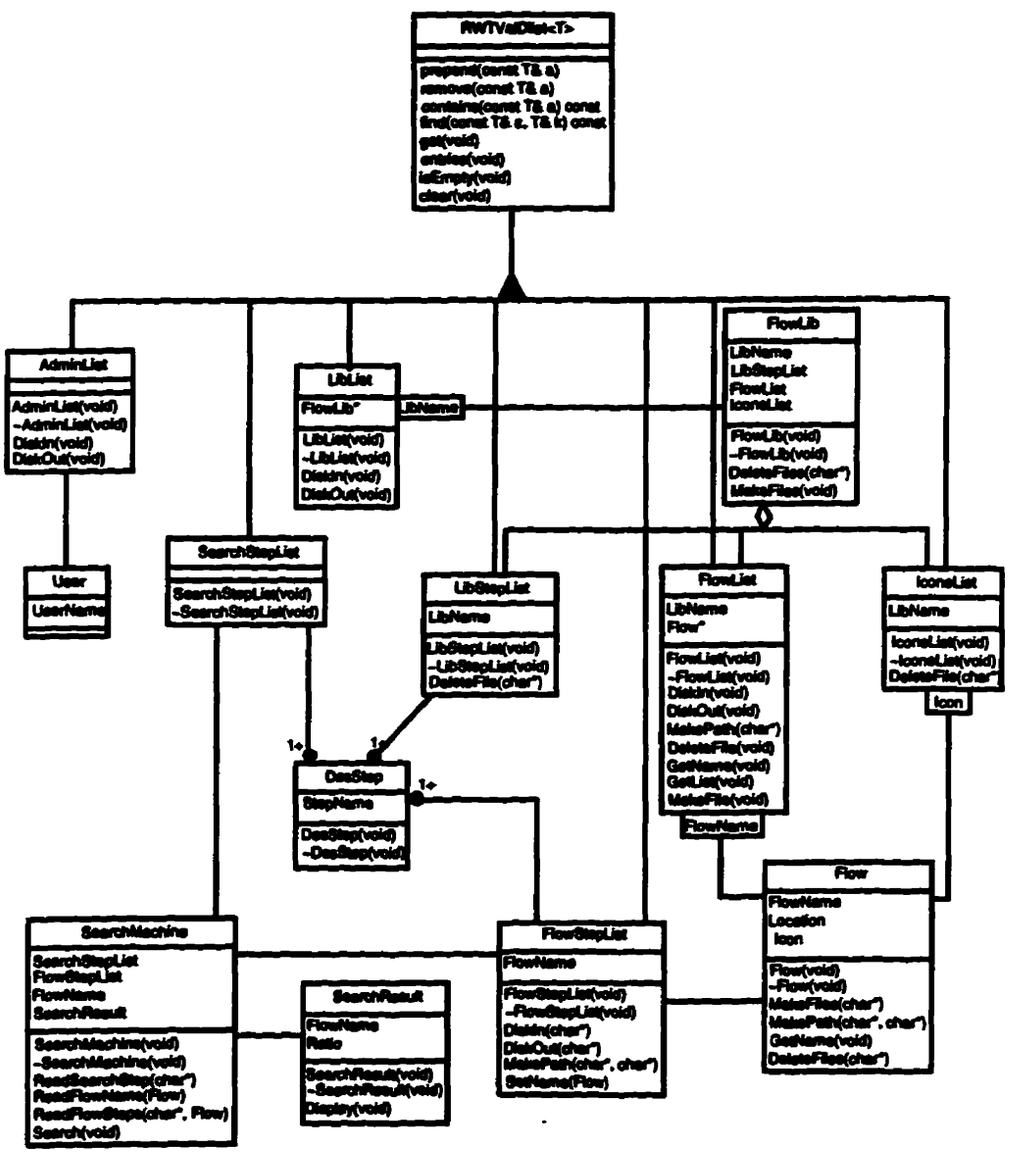


FIGURE 36. Object Model of the model subsystem of the Design Flow Manager with RWTValDlist<T> class template as the linked list superclass and class procedures and attributes as defined and used during the implementation.

3.3.2.3. Class attributes definition.

Class attributes which constitute member data are defined for each class at this step. Attributes are mainly a collection of member function arguments. Most of the attributes are hidden and this makes them accessible only by the member functions of the same class which is accomplished by defining them as private. Those which are to be accessed by external functions are defined as public.

The class attributes which have been extracted during the system analysis stage have been completed by new ones which are imposed by the implementation requirements (Figure 36).

3.4 Implementation.

Implementation has been organized into a number of separate phases. In the first phase the system code template has been generated. In the second phase the member functions have been implemented. The third phase focuses on the software control implementation for the particular use cases that leads to the creation of a set of separate compiled and scripting modules of code. The fourth phase focuses on the View implementation in the form of tcl/tk script. The fifth and last phase results in the implementation of Controller and software integration as expansion of the main script of the application [20].

3.4.1 Model implementation.

The Model has been implemented during the first three phases.

3.4.1.1. Code template.

Code template includes all the class declarations without the member function definitions. The resulting code represents the internal structure of the system. The code template has been generated with the help of "Software through Pictures" tool from Interactive Development Environments Inc. which has also been used during the system analysis and system design stages. As mentioned earlier, the C++ programming language has been used to implement the system [11][19][20]. The generated dfl.h header file is presented in Appendix A.

3.4.1.2. Member functions implementation.

The implementation of the member functions has been done manually and is presented in Appendix B as the dfl.cc file.

3.4.1.3. Use case compiled programs.

For some of the software menu options separate compiled programs have been implemented (Figure 37), which are presented in the Appendix C. Those menu options are:

- Grant User Administration Privileges,

- Revoke User Administration Privileges,

- Create New Library,

- Delete Library,

- Add New Flow,

- Delete Flow,

- Search Flow,

FIGURE 37. Executable programs for some of the menu options.

3.4.1.4. Use case tcl/tk implementation.

The remaining menu option actions have been implemented with the help of tcl/tk scripts and are included in the main application file dfl. Those menu options are as follows:

- New Flow,

- View Design Steps,

- Open Flow,

- List Existing Libraries,

- Open Library

3.4.2 View and Controller implementation.

View is completely implemented in the tcl/tk script as part of the dfl file (Appendix D). The main part of the Controller is also implemented in the same file. The remaining part of the Controller responsible for the argument passing mechanism is partially implemented in the C++ compiled programs.

The dfl file consists of the following main parts:

- main GUI components (View),
- messages (View),
- global variables (Controller),
- procedures (View & Controller),
- menu option procedures - means use cases (Controller & Model),
- GUI definition (View),

The main GUI components define the basic sections of the main application window. Messages are defined text variables used during application execution as entry field prompts or help information. A set of global variables has been defined to pass values between procedures and preserve basic application data such as application location in the "home" for example. Procedures have been defined to reuse some of the tasks, to organize the code and to implement support windows. Menu option procedures are implementations of the controlling part of the menu option execution. GUI definition implements the main application window.

4.0 Conclusions

In the first part of the thesis, the power of executable flows in design process automation has been presented. The PCB design and analysis flow developed here may be used even by inexperienced designers to ensure an accurate design process. Next, another level of design automation was proposed - the management of design flow libraries. The tool designed to provide the means for this management - Design Flow Manager - was presented. The Design Flow Manager may be a starting point to building a design expert system with more extended support for the designer.

The software was designed and implemented using the object-oriented approach. The methodology has been proven to be very effective in software development. The design process presentation and formulation starts from the early requirements statement and leads to the resulting implementation code. This is one of only a few available publications presenting the whole scope of the object-oriented software design process. Therefore, it is designated to provide guidelines to other designers considering employment of this methodology.

References and Bibliography

[1] P. Derrington, *Management of the Design Process, in Knowledge Based Expert Systems in Engineering: Planning and Design*, p. 19 - 33, Computational Mechanics Publications 1987.

[2] H.A. Simon, *Structure of Ill-Structured Problems*, Artificial Intelligence 4, p. 181-201, 1973.

[3] Mentor Graphics, *WorkXpert - WorkFlow Management Workshop*, Boston, Oct. 17-18, 1995.

[4] *Fablink User's Manual*, Mentor Graphics 1995.

[5] *PCB Products Overview Manual*, Mentor Graphics 1995.

[6] *Mastering Object Oriented Methods* - interactive tutorial, Action Software Inc., 1996.

[7] Ivar Jacobson, *Object-Oriented Software Engineering - A Use Case Driven Approach*, Addison-Wesley Publishing Company, 1992.

[8] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen, *Object-Oriented Modeling and Design*, Prentice-Hall Inc., 1991.

[9] Stuart Frost, *The Select Perspective - Developing Enterprise Systems using Object Technology*, Select Software Tools Inc., 1995.

[10] *Software through Pictures / Object Modeling Technique*, Interactive Development Environments Inc., 1995.

[11] Robert Lafore, *Object-Oriented programming in C++*, Waite Group Press, 1995.

[12] *Mastering Object Oriented Methods* - interactive tutorial, Action Software Inc., 1996.

[13] Ian Sommerville, *Software Engineering*, Addison-Wesley Publishing Company, 1992.

[14] Paul Godavari, *"Implementing MPEG audio code hardware through Rapid Prototyping Design Methodologies"* - Master Thesis under development, Department of Electrical and Computer Engineering, University of Manitoba.

[15] G.E.Krasner, S.T.Pope, A cookbook for using the model-view-controller user interface paradigm in Smalltalk-80, *Journal of Object-Oriented Programming*, 1(3):26-49, August/September 1988.

[16] E.Gamma, R.Helm, R.Johnson, J.Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, 1995, Addison Wesley Longman, Inc.

[17] John K. Ousterhout, "*Tcl and the Tk Toolkit*", Addison-Wesley Publishing Company, 1994.

[18] Thomas Keffer, "*Tools.h++. Foundation Class Library for C++ Programming - Version 6*". Rogue Wave Software Inc., 1994

[19] M.A. Ellis, B. Stroustrup, "*The Annotated C++ Reference Manual*", Addison-Wesley Publishing Company, 1995.

[20] H. Schildt, "*C++ the Complete Reference - Second Edition*", Osborne McGraw-Hill, 1995.

[20] S. McConnell, "*Code Complete A Practical Handbook / Software Construction*", Microsoft Press, 1993.

[21] A.D. Birrel, B.J. Nelson, "*Implementing remote procedure calls*". ACM Trans Computer Systems, vol. 2, pp. 39-59, 1984.

Appendix A.0 The header file dfl.h

```
// StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system flola_1

//Kris Kobylinski

//this is the dfl.h file


#ifndef _dfl_h_

#define _dfl_h_


#include <rw/tvdlist.h>

#include <rw/rstream.h>


// stp/omt class declarations

class LibStepList;

class User;

class AdminList;

class IconeList;

class SearchStepList;

class LibList;

class FlowStepList;

class FlowList;

class DesStep;

class Flow;

class FlowLib;

class SearchMachine;

class SearchResult;
```

```
// stp/omt class declarations end


// stp/omt class definition 39
class LibStepList :  public RWTValDlist<DesStep>
{
// stp/omt class members
public:
   LibStepList(void);
   LibStepList(char* c);
   ~LibStepList(void);
   void DeleteFile(char*); //REMOVES THE LIST FILE
protected:
   DesStep ptrDesStep;
   //DesStep contains;
   //LibStepListIterator ptrLibStepListIterator;
   FlowLib ptrFlowLib;
private:
   char* LSLLibName;
// stp/omt class members end
};
// stp/omt class definition end


class User
{
public:
```

```
  User(void);
  ~User(void);
  User(char* c);
  void User( const User& u);
 //define an assignment operator:
 void operator=(const User& u);
  //define an equality test operator:
 int operator==(const User& u)const;
 friend ostream& operator <<(ostream& str, User& u);
protected:
private:
  char* UserName;
};

// stp/omt class definition 21
class AdminList : public RWTValDlist<User>
{
// stp/omt class members
public:
   AdminList(void);
   ~AdminList(void);
   void DiskIn(void); //READS THE LIST FROM FILE
   void DiskOut(void); //WRITES THE LIST TO FILE
protected:
   User ptrUser;
```

```
    //User contains;
    //AdminListIterator ptrAdminListIterator;
  private:
  // stp/omt class members end
  };
  // stp/omt class definition end

  // stp/omt class definition 29
  class IconeList : public RWTValDlist<char>
  {
  // stp/omt class members
  public:
    IconeList(void);
    IconeList(char* c);
    void ~IconeList(void);
    void DeleteFile(char*); //REMOVES THE LIST FILE
  protected:
    Flow ptrFlow;
    Flow keepsiconein;
    //IconeListIterator ptrIconeListIterator;
    FlowLib ptrFlowLib;
  private:
    char* ILLibName;
  // stp/omt class members end
  };
```

```
// stp/omt class definition end


// stp/omt class definition 27
class SearchStepList : public RWTValDlist<DesStep>
{
// stp/omt class members
public:
   SearchStepList(void);
   ~SearchStepList(void);
protected:
   DesStep ptrDesStep;
   SearchMachine ptrSearchMachine;
   DesStep contains;
   //SearchStepListIterator ptrSearchStepListIterator;
private:
// stp/omt class members end
};
// stp/omt class definition end


// stp/omt class definition 1
class LibList : public RWTValDlist<FlowLib>
{
// stp/omt class members
public:
   LibList(void);
```

```
    ~LibList(void);
    void DiskIn(void); //READS THE LIST FROM FILE
    void DiskOut(void); //WRITES THE LIST TO FILE
  protected:
    FlowLib ptrFlowLib;
    FlowLib contains;
    //LibLIstIterator ptrLibLIstIterator;
  private:
  // stp/omt class members end
  };
  // stp/omt class definition end

  // stp/omt class definition 59
  class FlowStepList : public RWTValDlist<DesStep>
  {
  // stp/omt class members
  public:
    FlowStepList(void);
    FlowStepList(char* c);
    ~FlowStepList(void);
    void DiskIn(char*); //READS THE LIST FROM FILE
    void DiskOut(char*); //WRITES THE LIST TO FILE
    char* MakePath(char*, char*); //RETURNS THE PATH TO FILE
    void SetName(Flow); //SETS FSLFlowName VALUE
  protected:
```

```
    Flow ptrFlow;
    DesStep ptrDesStep;
    SearchMachine ptrSearchMachine;
    Flow has;
    SearchMachine accesses;
    DesStep contains;
    //FlowStepListIterator ptrFlowStepListIterator;
private:
    char* FSLFlowName;
// stp/omt class members end
};
// stp/omt class definition end

// stp/omt class definition 16
class FlowList : public RWTValDlist<Flow>
{
// stp/omt class members
public:
    FlowList(void);
    FlowList(char* c);
   ~FlowList(void);
   void DiskIn(void); //READS THE LIST FROM FILE
   void DiskOut(void); //WRITES THE LIST TO FILE
   char* MakePath(char*); //RETURNS THE PATH TO FILE
   void DeleteFile(void); //REMOVES THE LIST FILE
```

```
    void GetName(void); //RETURNS THE FLLibName VALUE
    void GetList(void); //RETURNS THE LIST
    void MakeFile(void); //CREATES THE LIST FILE
protected:
    Flow ptrFlow;
    Flow contains;
    //FlowListIterator ptrFlowListIterator;
    FlowLib ptrFlowLib;
private:
    char* FLLibName;
// stp/omt class members end
};
// stp/omt class definition end

// stp/omt class definition 63
class DesStep
{
// stp/omt class members
public:
    DesStep(void);
    DesStep(char* c);
  ~DesStep();
  DesStep( const DesStep& ds);
   //define an assignment operator:
 void operator=(const DesStep& ds);
```

```
 //define an equality test operator:
 int operator==(const DesStep& ds)const;
 friend ostream& operator <<(ostream& str, DesStep& ds);
protected:
   FlowStepList ptrFlowStepList;
   LibStepList ptrLibStepList;
   SearchStepList ptrSearchStepList;
   LibStepList contains;
   SearchStepList contains;
   FlowStepList contains;
private:
   char* StepName;
// stp/omt class members end
};
// stp/omt class definition end


// stp/omt class definition 51
class Flow
{
// stp/omt class members
public:
    Flow(void);
   Flow(char*);
   Flow(char*, char*, char*);
   ~Flow(void);
```

```
    void MakeFiles(char*); //CREATES THE FLOW INFORMATION FILES
    char* MakePath(char*, char*); //RETURNS THE PATH TO THE FILES
    char* GetName(void); //RETURNS THE FlowName VALUE
    void DeleteFiles(char*); //REMOVES THE FLOW INFORMATION FILES
    //define an assignment operator:
    void operator=(const Flow& f);
    //define an equality test operator:
    int operator==(const Flow& f)const;
    friend ostream& operator <<(ostream& str, Flow& f);
  protected:
    FlowList ptrFlowList;
    IconeList ptrIconeList;
    FlowStepList ptrFlowStepList;
    FlowList contains;
    IconeList keepsiconein;
    FlowStepList has;
  private:
    char* FlowName;
    char* Location;
    char* Icone;
  // stp/omt class members end
  };
  // stp/omt class definition end

  // stp/omt class definition 10
```

```
class FlowLib
{
// stp/omt class members
public:
   FlowLib(void);
   FlowLib(char* c);
   ~FlowLib(void);
   FlowLib( const FlowLib& fl);
   void DeleteFiles(char*); //REMOVES THE FILES
   void MakeFiles(void); //CREATES THE FILES
   //define an assignment operator:
   void operator=(const FlowLib& fl);
   //define an equality test operator:
   int operator==(const FlowLib& fl)const;
   friend ostream& operator <<(ostream& str, FlowLib& fl);
protected:
   LibList ptrLibList;
   LibList contains;
   FlowList flist;
   LibStepList lslist;
   IconeList ilist;
private:
   char* LibName;
// stp/omt class members end
};
```

```
// stp/omt class definition end


// stp/omt class definition 71
class SearchMachine
{
// stp/omt class members
public:
   SearchMachine(void);
   ~SearchMachine(void);
   void ReadSearchStep(char*); //READS THE LIST
   void ReadFlowName(Flow); //READS THE VALUE OF THE NEXT FLOW NAME FROM THE LIST
   void ReadFlowSteps(char*, Flow); //READS THE LIST
   float Search(void); //RETURNS A RESULT OF STEPS COMPARISON
protected:
   FlowStepList ptrFlowStepList;
   SearchStepList ptrSearchStepList;
   SearchResult ptrSearchResult;
   SearchResult generates;
   FlowStepList accesses;
private:
// stp/omt class members end
};
// stp/omt class definition end


// stp/omt class definition 66
```

```
class SearchResult
{
// stp/omt class members
public:
   SearchResult(void);
   ~SearchResult(void);
   SearchResult(Flow f, float r);
   void Display(void); //DISPLAY THE Ratio VALUE
protected:
   SearchMachine ptrSearchMachine;
   SearchMachine generates;
private:
   char* FlowName;
   float Ratio;
// stp/omt class members end
};
// stp/omt class definition end


// stp/omt footer
#endif
// stp/omt footer end
```

Appendix B.0 The member functions implementation file dfl.cc

```
////////////////////////////////////////
//StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system flola_1
//Kris Kobylinski
//this is the dfl.cc file with member function definitions.

#include "dfl.h"
#include <rw/tvdlist.h>
#include <rw/rstream.h>
#include <string.h>
#include <iostream.h>
#include <fstream.h>
#include <stream.h>

extern "C" {
#include <stdio.h>
int remove_file(const char *path);
}

#define FN "lib.names"
#define FL "flow_list.names"
#define LSL "lib_step_list.names"
#define IL "icone_list.names"
#define FSL "flow_step_list.names"
#define FLOC "location.dfl"
#define ILOC "icone.dfl"
#define USERS "users.dfl"
#define SLASH "/"
int STR_LEN = 30;
int USER_NAME = 8;

int remove_file(const char *path)
{
```

```
remove(path);
return 0;
}

//////////LibStepList//////////
// stp/omt operation 39::110

LibStepList::LibStepList(){
}
// stp/omt operation end
LibStepList::LibStepList(char* c) {
 //set the LSLLibName attribute
 LSLLibName = new char[strlen(c) + 1];
 strcpy(LSLLibName, c);
}
// stp/omt operation 39::111

LibStepList::~LibStepList(){
 delete [] LSLLibName;
}
// stp/omt operation end

// stp/omt operation 39::113
void
LibStepList::DeleteFile(char* c) {
 //initialize the flow_list.names file
 char* dfl = new char[strlen(c)+1];
 strcpy(dfl, c);
 char* efl = SLASH;
 strcat(dfl,efl);
 efl = LSL;
 strcat(dfl, efl);
 const char* path = dfl;
```

```
  remove_file(path);
  delete dfl;
}
// stp/omt operation end

//////////////User/////////////////
////////////////////////////////////////////////////
User::User() {
}

User::User(char*c) {
  UserName = new char[USER_NAME];
  strcpy(UserName, c);
}

User::~User() {
}

User::User( const User& u) {
   UserName = new char[strlen(u.UserName)+1];
   strcpy(UserName, u.UserName);
 }

void
User::operator=(const User& u) {
    if ( this != &u) {
delete UserName;
    UserName = new char[strlen(u.UserName)+1];
    strcpy(UserName, u.UserName);
  }
 }

 //define an equality test operator:
```

```
int
User::operator==(const User& u) const {
   return strcmp(UserName, u.UserName)==0;
 }

ostream&
operator <<(ostream& str, User& u) {
   str << u.UserName;
   return str;
}

////////////AdminList/////////////
// stp/omt operation 21::107

AdminList::AdminList(){
}
// stp/omt operation end

// stp/omt operation 21::108

AdminList::~AdminList(){
}
// stp/omt operation end

// stp/omt operation 21::136
void
AdminList::DiskIn() {
 //this puts the UserNames from the USERS file into AdminList
 const char* path = new char[STR_LEN];
 path = USERS;
 char* ch_ptr = new char[STR_LEN];
 char* test = new char[2];
 test = "\0\0";
```

```
  char d;
  int i=0;
  fstream infile;
  infile.open(path, ios::in|ios::binary);
  //if(infile) {
   while(!infile.eof()) {
    infile.seekg(i, ios::beg);
    infile >> d;
    if(!infile.eof()) {
if(d != '#') {
 *test = d;
 strcat(ch_ptr, test);
 i++;
}
else {
 append(ch_ptr);
 *ch_ptr = '\0';
 i+=2;
}
    }
   }
   //}
   //delete [] ch_ptr;
}

// stp/omt operation end

// stp/omt operation 21::148
void
AdminList::DiskOut() {
 //this puts the AdminList into the USERS file
 const char* path = new char[STR_LEN];
 path = USERS;
```

```
  remove_file(path);
 fstream outfile;
 outfile.open(path, ios::outlios::binary);
 while(!isEmpty())
  outfile << get() << "#" << endl;
}

// stp/omt operation end

////////////////IconeList////////////
// stp/omt operation 29::137

IconeList::IconeList(){
}
// stp/omt operation end

IconeList::IconeList(char* c) {
 //set the ILLibName attribute
 ILLibName = new char[strlen(c) +1];
 strcpy(ILLibName, c);
}

// stp/omt operation 29::138

IconeList::~IconeList(){
 delete [] ILLinName;
}
// stp/omt operation end

// stp/omt operation 29::139
void
IconeList::DeleteFile(char* c) {
```

```
  //initialize the flow_list.names file
  char* dfl = new char[strlen(c)+1];
  strcpy(dfl, c);
  char* efl = SLASH;
  strcat(dfl,efl);
  efl = IL;
  strcat(dfl, efl);
  const char* path = dfl;
  cout << "path =" << path << endl;//test
  remove_file(path);
  delete dfl;
}
// stp/omt operation end

////////////SearchStepList//////////////
// stp/omt operation 27::95

SearchStepList::SearchStepList(){
}
// stp/omt operation end

// stp/omt operation 27::96

SearchStepList::~SearchStepList(){
}
// stp/omt operation end

////////////LibList/////////////////
LibList::LibList() {
}

LibList::~LibList() {
}
```

```
void
LibList::DiskIn() {
 //this puts the UserNames from the USERS file into AdminList
 const char* path = new char[STR_LEN];
 path = FN;
 char* ch_ptr = new char[STR_LEN];
 char* test = new char[2];
 test = "\0\0";
 char d;
 int i=0;
 fstream infile;
 infile.open(path, ios::in|ios::binary);
 if(infile) {
  while(!infile.eof()) {
   infile.seekg(i, ios::beg);
   infile >> d;
   if(!infile.eof()) {
if(d != '#') {
 *test = d;
 strcat(ch_ptr, test);
 i++;
}
else {
 append(ch_ptr);
 *ch_ptr = '\0';
 i+=2;
}
   }
  }
 }
  //delete [] ch_ptr;
}
```

```
void
LibList::DiskOut() {
 //this puts the LibList into the lib.name file
 const char * path = FN;
 remove_file(path);
 fstream outfile;
 outfile.open(path, ios::outlios::binary);
 while(!isEmpty())
  outfile << get() << "#" << endl;
}

////////////FlowStepList////////////////
// stp/omt operation 59::114

FlowStepList::FlowStepList(){
}
// stp/omt operation end

FlowStepList::FlowStepList(char* c) {
 FSLFlowName = new char[strlen(c) +1];
 strcpy(FSLFlowName, c);
}

// stp/omt operation 59::115

FlowStepList::~FlowStepList(){
 delete [] FSLFlowName;
}
// stp/omt operation end

// stp/omt operation 59::116
void
```

```
FlowStepList::DiskOut(char* c) {
 //this puts the FlowList into the LIB_NAME/FL file
 const char* path = new char[STR_LEN];
 path = MakePath(c, FSL);
 remove_file(path);
 fstream outfile;
 outfile.open(path, ios::outlios::binary);
 while(!isEmpty())
  outfile << get() << "#" << endl;
}
// stp/omt operation end

// stp/omt operation 59::167
char*
FlowStepList::MakePath(char *c, char* d) {
 //this procedure returns the path to the file LIB_NAME/FLOW_NAME/FSL
 //c is LibName, d is FSL
 char* dfl = new char[STR_LEN];
 strcpy(dfl, c);
 char* efl = new char[STR_LEN];
 efl =SLASH;
 strcat(dfl,efl);
 strcat(dfl, FSLFlowName);
 strcat(dfl,efl);
 efl = d;
 strcat(dfl, efl);
 return dfl;
}
// stp/omt operation end

// stp/omt operation 59::168
void
FlowStepList::SetName(Flow f) {
```

```
  FSLFlowName = new char[STR_LEN];
  strcpy(FSLFlowName, f.GetName());
}
// stp/omt operation end

// stp/omt operation 59::169
void
FlowStepList::DiskIn(char* c) {
  //this puts the StepNames from the FSL file into FlowStepList
  const char* path = new char[STR_LEN];
  path =MakePath(c, FSL);
    char* ch_ptr = new char[STR_LEN];
    char* test = new char[2];
    test = "\0\0";
    char d;
    int i=0;
    fstream infile;
    infile.open(path, ios::in|ios::binary);
    while(!infile.eof()) {
      infile.seekg(i, ios::beg);
      infile >> d;
      if(!infile.eof()) {
      if(d != '#') {
*test = d;
strcat(ch_ptr, test);
i++;
      }
      else {
DesStep* ds_ptr = new DesStep(ch_ptr);
append(*ds_ptr);
//delete ds_ptr;
*ch_ptr = '\0';
i+=2;
```

```
    }
    }
   }
   //delete [] ch_ptr;
}
// stp/omt operation end

////////////////FlowList/////////////
// stp/omt operation 16::100

FlowList::FlowList(){
}
// stp/omt operation end

// stp/omt operation 16::101

FlowList::~FlowList(){
}
// stp/omt operation end

// stp/omt operation 16::102
FlowList::FlowList(char* c) {
 //set the FLLibName attribute
 FLLibName = new char[strlen(c) +1];
 strcpy(FLLibName, c);
}
// stp/omt operation end

// stp/omt operation 16::156
void
FlowList::MakeFile() {

 //initialize the flow_list.names file
```

```
  const char* path  = new char[STR_LEN];
  path = MakePath(FL);
  fstream outfile;
  outfile.open(path, ios::outlios::binarylios::beg);
  outfile << '#' << endl;
  outfile.close();
}
// stp/omt operation end

// stp/omt operation 16::157
void
FlowList::DeleteFile() {
  //initialize the flow_list.names file
  const char* path = new char[STR_LEN];
  path = MakePath(FL);
  remove_file(path);
}
// stp/omt operation end

// stp/omt operation 16::158
void
FlowList::DiskIn() {
  //this puts the FlowNames from the FL file into FlowList
  const char* path = new char[STR_LEN];
  path =MakePath(FL);
    char* ch_ptr = new char[STR_LEN];
    char* test = new char[2];
    test = "\0\0";
    char d;
    int i=0;
    ifstream infile;
    infile.open(path, ios::inlios::binary);
    while(!infile.eof()) {
```

```
    infile.seekg(i, ios::beg);
    infile >> d;
    if(!infile.eof()) {
    if(d != '#') {
*test = d;
strcat(ch_ptr, test);
i++;
    }
    else {
Flow* flow_ptr = new Flow(ch_ptr);
append(*flow_ptr);
delete flow_ptr;
*ch_ptr = '\0';
i+=2;
    }
    }
  }
  delete [] ch_ptr;
  //delete [] d;
}
// stp/omt operation end

// stp/omt operation 16::159
void
FlowList::DiskOut() {
 //this puts the FlowList into the LIB_NAME/FL file

 const char* path = new char[STR_LEN];
 fstream outfile;
 path = MakePath(FL);
 remove_file(path);
 outfile.open(path, ios::out|ios::binary|ios::app);
 while(!isEmpty())
```

```
    outfile << get() << "#" << endl;
}
// stp/omt operation end
char*
FlowList::MakePath(char* c) {
 //this procedure returns the path to the file
 char* dfl = new char[STR_LEN];
 strcpy(dfl, FLLibName);
 char* efl =new char[STR_LEN];
 efl =SLASH;
 strcat(dfl,efl);
 efl = c;
 strcat(dfl, efl);
 //delete [] efl;
 return dfl;
}

char* FlowList::GetName() {
return FLLibName;
}

////////////////DesStep//////////////////////
// stp/omt operation 63::117

DesStep::DesStep(){
}
// stp/omt operation end
DesStep::DesStep(char* c) {
 StepName = new char[strlen(c) +1];
 strcpy(StepName, c);
}
// stp/omt operation 63::118
```

```
DesStep::~DesStep(){
  delete [] StepName;
}
// stp/omt operation end
DesStep::DesStep( const DesStep& ds) {
   StepName = new char[strlen(ds.StepName)+1];
   strcpy(StepName, ds.StepName);
 }

void
DesStep::operator=(const DesStep& ds) {
    if ( this != &ds) {
delete StepName;
    StepName = new char[strlen(ds.StepName)+1];
    strcpy(StepName, ds.StepName);
  }
 }

int
DesStep::operator==(const DesStep& ds) const {
   return strcmp(StepName, ds.StepName)==0;
}

ostream&
operator <<(ostream& str, DesStep& ds) {
   str << ds.StepName;
   return str;
 }

//////////////Flow/////////////////
// stp/omt operation 51::140

Flow::Flow(){
```

```
}
// stp/omt operation end
Flow::Flow(char* c) {
 FlowName = new char[strlen(c) +1];
 strcpy(FlowName, c);
}

Flow::Flow(char* c, char* d, char* e) {
 FlowName = new char[strlen(c) + 1];
 strcpy(FlowName, c);
 Location = new char[strlen(d) +1];
 strcpy(Location, d);
 Icone = new char[strlen(e) +1];
 strcpy(Icone, e);
}
// stp/omt operation 51::141

Flow::~Flow(){
 delete [] FlowName;
}
// stp/omt operation end
Flow::Flow( const Flow& f) {
   FlowName = new char[strlen(f.FlowName)+1];
   strcpy(FlowName, f.FlowName);
 }
// stp/omt operation 51::142
char*
Flow::MakePath(char *c, char* d) {
 //this procedure returns the path to the file LIB_NAME/FLOW_NAME/FLOC or ILOC
 //c is LibName, d is FLOC or ILOC
 char* dfl = new char[STR_LEN];
 strcpy(dfl, c);
 char* efl = new char[STR_LEN];
```

```
  efl =SLASH;
  strcat(dfl,efl);
  strcat(dfl, FlowName);
  strcat(dfl,efl);
  efl = d;
  strcat(dfl, efl);
  return dfl;
}

// stp/omt operation end

// stp/omt operation 51::143
void
Flow::MakeFiles(char* c) {
  //c is LibName
  const char* path = new char[STR_LEN];
  path = MakePath(c, FLOC);
  remove_file(path);
  fstream outfile;
  outfile.open(path, ios::outlios::binary);
  outfile << Location << endl;
  outfile.close();

  path = MakePath(c, ILOC);
  remove_file(path);
  outfile.open(path, ios::outlios::binary);
  outfile << Icone << endl;
  outfile.close();
}
// stp/omt operation end

// stp/omt operation 51::144
char*
```

```
Flow::GetName() {
  return FlowName;
}
// stp/omt operation end

void
Flow::operator=(const Flow& f) {
  //if ( this != &f) {
//delete FlowName;
//delete Location;
//delete Icone;
//FlowName = new char[strlen(f.FlowName)+1];
FlowName = new char[STR_LEN];
    strcpy(FlowName, f.FlowName);
    //Location = new char[strlen(f.Location)+1];
    //Location = new char[STR_LEN];
    //strcpy(Location, f.Location);
    //Icone = new char[strlen(f.Icone)+1];
    //Icone = new char[STR_LEN];
    //strcpy(Icone, f.Icone);
    //}
  }

int
Flow::operator==(const Flow& f) const {
   return strcmp(FlowName, f.FlowName)==0;
}

ostream&
operator <<(ostream& str, Flow& f) {
   str << f.FlowName;
   return str;
  }
```

```
//////////////////FlowLib//////////////////////
// stp/omt operation 10::145

FlowLib::FlowLib(): flist(), lslist(), ilist() {
}
// stp/omt operation end

FlowLib::FlowLib(char* c):flist(c), lslist(c), ilist(c) {
   LibName = new char[strlen(c) +1];
   strcpy(LibName, c);
 }
// stp/omt operation 10::146

FlowLib::~FlowLib(){
 delete [] LibName;
}
// stp/omt operation end

// stp/omt operation 10::147
FlowLib::FlowLib( const FlowLib& fl) {
   LibName = new char[strlen(fl.LibName)+1];
   strcpy(LibName, fl.LibName);
 }
// stp/omt operation end
void
FlowLib::MakeFiles() {
 //calls make file procedure of FlowLib components
 flist.MakeFile();
 //lslist.MakeFile(c);
 //ilist.MakeFile(c);
}
```

```
void
FlowLib::DeleteFiles(char* c) {
 flist.DeleteFile();
 lslist.DeleteFile(c);
 ilist.DeleteFile(c);
 remove_file(c);
}

char*
FlowLib::GetName() {
return LibName;
}

void
FlowLib::operator=(const FlowLib& fl) {
    if ( this != &fl) {
delete LibName;
    LibName = new char[strlen(fl.LibName)+1];
    strcpy(LibName, fl.LibName);
  }
 }

int
FlowLib::operator==(const FlowLib& fl) const {
  return strcmp(LibName, fl.LibName)==0;
 }

ostream&
operator <<(ostream& str, FlowLib& fl) {
  str << fl.LibName;
  return str;
 }
```

```
////////////////SearchMachine/////////////
SearchMachine::SearchMachine() : ssl(), fsl(), sr() {
}

SearchMachine::~SearchMachine() {
}

void
SearchMachine::ReadSearchStep(char* ar) {
   ssl.append(ar);
}

void
SearchMachine::ReadFlowName(Flow f) {
 FlowName = new char[STR_LEN];
 strcpy(FlowName, f.GetName());
}

void
SearchMachine::ReadFlowSteps(char* c, Flow f) {
 fsl.SetName(f);
 fsl.clear();
 fsl.DiskIn(c);
}

float
SearchMachine::Search() {
 int no_ssteps = 0;
 int to_ssteps = ssl.entries();
 for(int i = 0; i < to_ssteps; i++) {
  if(fsl.contains(ssl.at(i)))
    no_ssteps++;
 }
```

```
  return (float) no_ssteps / (float) to_ssteps;
}

void
SearchMachine::TestDisplay() {
  while(!fsl.isEmpty())
  cout << fsl.get() << " flowstep " << endl;
}
////////////////SearchResult/////////////////
// stp/omt operation 66::119

SearchResult::SearchResult(){
}
// stp/omt operation end

SearchResult::SearchResult(Flow f, float r) {
  FlowName = new char[STR_LEN];
  strcpy(FlowName, f.GetName());
  Ratio = r;
}

// stp/omt operation 66::120
SearchResult::~SearchResult(){
}
// stp/omt operation end

// stp/omt operation 66::121
void
SearchResult::Display() {
  cout << FlowName << " has ";
  cout << (int)(Ratio*100) << "% of required steps" << endl;
}
// stp/omt operation end
```

Appendix C.0 The C++ implemented menu option files.

C.1 File addadmin.cc

```
//Kris Kobylinski 08/7/97
//mono file addadmin.cc for the Add New Flow menu option.
////////////////////////////////////////
//Pre:
//
////////////////////////////////////////
//Input:
//1: your UserName
//2: new UserName
////////////////////////////////////////
//Function:
//1. Reads the AdminList from the users.dfl file
//2. Checks if the UserName is not included in the AdminList
//3. Adds the UserName
//4. Writes back the AdminList
////////////////////////////////////////
//To implement:


inlcude "dfl.h"


int
main(int argc, char** argv)
{
```

```
//command line arguments:
//1: your UserName
//2: new UserName
if(argc >= 3)
 {
  AdminList al;
  al.DiskIn();
  User u(argv[1]);
  User him(argv[2]);
  if(al.contains(u)) {
   if(al.contains(him)) {
      cout << "The user has already the administration priviledges granted." << endl;
      return 0;
      }
   else {
      al.append(argv[2]);
      al.DiskOut();
      return 0;
      }
  }
else {
   cout << "ERROR: You are not allowed to perform this operation." << endl;
   return 0;
   }
}
```

```
 else {
   cout << "ERROR: You did not enter user ID!" << endl;
   return 0;
   }
}
```

C.2 File deladmin.cc

```
// StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system fiola_1
//Kris Kobylinski 08/7/97
//mono file deladmin.cc for the Revoke User Administrative Priviledges  menu option.
////////////////////////////////////////////
//Pre:
//
////////////////////////////////////////////
//Input:
//1: your UserName
//2: absolete UserName
////////////////////////////////////////////
//Function:
//1. Reads the AdminList from the users.dll file
//2. Checks if the UserName is not included in the AdminList
//3. Removes the UserName
//4. Writes back the AdminList
////////////////////////////////////////////
//To implement:
```

```
#include "dfl.h"
 int
main(int argc, char** argv)
{
 //command line arguments:
 //1: your UserName
 //2: his UserName
 if(argc >= 3)
  {
   AdminList al;
   al.DiskIn();
   User u(argv[1]);
   User him(argv[2]);
   if(al.contains(u)) {
     if(!al.contains(him)) {
       cout << "The user has not been granted the administration priviledges." << endl;
       return 0;
       }
    else {
      al.remove(argv[2]);
      al.DiskOut();
      return 0;
     }
   }
   else {
```

```
    cout << "ERROR: You are not allowed to perform this operation!" << endl;
    return 0;
   }
  }
 else {
  cout << "ERROR: You did not enter user ID!" << endl;
  return 0;
 }
}
```

C.3 File newlib.cc

```
// StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system flola_1
//Kris Kobylinski 08/8/97
//mono file newlib.cc for the New Library menu option.
////////////////////////////////////////
//Pre:
//1. Create the LibName directory
////////////////////////////////////////
//Input:
//1: UserName
//2: LibName
////////////////////////////////////////
//Function:
//1. Checks the administration priviledges of the user
//2. Reads the LibList from the FN file
//3. Checks if the LibName is not included in the LibList
```

```
//4. Adds the LibName
//5. Creates the LibName/FL file
//6. Writes back the LibList
////////////////////////////////////////
//To implement:

#include "dfl.h"

int
main(int argc, char** argv)
{
 //command line arguments:
 //1: UserName
 //2: LibName
 if(argc >= 3)
 {
   AdminList al;
   al.DiskIn();
   User u(argv[1]);
   if(al.contains(u)) {
     LibList ll;
     ll.DiskIn();
     FlowLib fl(argv[2]);
     if(!ll.contains(fl)) {
       ll.append(argv[2]);
```

```
            fl.MakeFiles();
            fl.DiskOut();
            return 0;
            }
          else {
            cout << argv[2] << " library is already created ." <<endl;
            return 0;
            }
        }
        else {
            cout << "ERROR: You are not allowed to perform the task." << endl;
            return 0;
           }
      }
     else {
        cout << "ERROR: You did not enter new library name!" << endl;
        return 0;
       }
}
```

C.4 File dellib.cc

```
// StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system flola_1
//Kris Kobylinski 08/8/97
//mono file dellib.cc for the Delete Library menu option.
////////////////////////////////////////
//Pre:
```

```
////////////////////////////////////////////

//Input:

//1: UserName

//2: LibName

////////////////////////////////////////////

//Function:

//1. Checks the administration priviledges of the user

//2. Reads the LibList from the FN file

//3. Checks if the LibName is not included in the LibList

//4. Deletes the LibName

//5. Deletes the LibName/FL file

//6. Writes back the LibList

////////////////////////////////////////////

//To implement:


#include "dfl.h"


 int

main(int argc, char** argv)

{

 //command line arguments:

 //1: UserName

 //2: LibName


 if(argc >= 2)
```

```
{
  AdminList al;
  al.DiskIn();
  User u(argv[1]);
  if(al.contains(u)) {
    LibList ll;
    ll.DiskIn();
    FlowLib fl(argv[2]);
    if(ll.contains(fl)) {
      ll.remove(argv[2]);
      fl.DeleteFiles(argv[2]);
      ll.DiskOut();
      return 0;
      }
   else {
      cout << "ERROR: " << argv[2] << " is not a valid library name." << endl;
      return 0;
      }
  }
  else {
    cout << "ERROR: You are not allowed to perform the task." << endl;
    return 0;
    }
}
else {
```

```
        cout << "ERROR: You did not enter new library name!" << endl;
        return 0;
      }
}
```

C.5 File newflow.cc

```
// StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system flola_1
//Kris Kobylinski 07/28/97
//mono file newflow.cc for the Add New Flow menu option.
////////////////////////////////////////
//Pre:
//Create the LIB_NAME/FLOW_NAME directory
////////////////////////////////////////
//Input:
//1: UserName
//2: LibName
//3: FlowName
//4: Location
//5: Icone location
//6...:FlowSteps
////////////////////////////////////////
//Function:
//1. Checks user admin priv
//2. Adds new FlowName to LIB_NAME/FL file
//3. Creates LIB_NAME/FLOW_NAME/flow_step_list.names and fills with the flow's steps
//4. Creates LIB_NAME/FLOW_NAME/FLOC and puts there Location
```

```
//5. Creates LIB_NAMES/FLOW_NAME/ILOC and puts there flow's icone location or puts default one.
////////////////////////////////////////

#include "dfl.h"

int
main(int argc, char** argv)
{
 //command line arguments:
 //1: UserName
 //2: LibName
 //3: FlowName
 //4: Location
 //5: Icone location
 //6...:FlowSteps
 if(argc >= 7)
 {
   AdminList al;
   al.DiskIn();
   User u(argv[1]);
   if(al.contains(u)) {
     FlowList fl(argv[2]);
     fl.DiskIn();
     fl.append(argv[3]);
     fl.DiskOut();
```

```
        FlowStepList fsl(argv[3]);
        for(int i=6; i<=argc; i++)
          fsl.append(argv[i]);
        fsl.DiskOut(argv[2]);
        Flow f(argv[3],argv[4],argv[5]);
        f.MakeFiles(argv[2]);
        return 0;
       }
       else {
         cout << "ERROR: You are not allowed to perform this operation." << endl;
         return 0;
         }
   }
  else {
     cout << "ERROR: The flow data is inclomlete!" << endl;
     return 0;
     }
}
```

C.6 File delflow.cc

```
// StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system fiola_1
//Kris Kobylinski 07/31/97
//mono file delflow.cc for the Delete Flow menu option.
////////////////////////////////////////////
//Pre:
//
```

```
////////////////////////////////////////

//Input:

//1: UserName

//2: LibName

//3: FlowName

////////////////////////////////////////

//Function:

//1. Checks user admin privs

//2. Deletes FlowName from LIB_NAME/FL file

//3. Deletes file LIB_NAME/FLOW_NAME/flow_step_list.names

//4. Deletes file LIB_NAME/FLOW_NAME/FLOC

//5. Deletes files LIB_NAMES/FLOW_NAME/ILOC

//6. Deletes directory LIB_NAMES/FLOW_NAME

////////////////////////////////////////


#include "dfl.h"


 int

main(int argc, char** argv)

{

 //command line arguments:

 //1: UserName

 //2: LibName

 //3: FlowName

 if(argc >= 4)
```

```
{
  AdminList al;
  al.DiskIn();
  User u(argv[1]);
  if(al.contains(u)) {
    FlowList fl(argv[2]);
    fl.DiskIn();
    fl.remove(argv[3]);
    fl.DiskOut();
    Flow f(argv[3]);
    f.DeleteFiles(argv[2]);
    return 0;
    }
  else {
    cout << "ERROR: You are not allowed to perform this operation." << endl;
    return 0;
    }
  }
  else {
    cout << "ERROR: You did not enter the flow name!" << endl;
    return 0;
    }
}
```

C.7 File search.cc

```
// StP -- created on Thu Jul 17 12:52:02 1997 for kobylin@brandy from system flola_1
```

```
//Kris Kobylinski 08/4/97
//mono file searchflow.cc for the Search Flow menu option.
////////////////////////////////////////
//Pre:
//
////////////////////////////////////////
//Input:
//1: LibName
//2...:SearchSteps
////////////////////////////////////////
//Function:
//1.
////////////////////////////////////////
//To implement:

#include "dfi.h"

 int
main(int argc, char** argv)
{
 //command line arguments:
 //1: LibName
 //2...: SearchSteps

 if(argc >= 3)
```

```
  {
   FlowList fl(argv[1]);
   fl.DiskIn();
   SearchMachine sm;
   cout << "Chosen design steps: ";
   for(int i=2; i<argc; i++) {
     cout <<  argv[i]<< " ";
     sm.ReadSearchStep(argv[i]);
     }
   cout << endl;
   Flow f;
   while(!fl.isEmpty()) {
     f =fl.get();
     sm.ReadFlowName(f);
     sm.ReadFlowSteps(argv[1], f);
     SearchResult sr(f, sm.Search());
     sr.Display();
     }
   return 0;
  }
  else {
     cout << "ERROR: You did not choose any design steps !" << endl;
     return 0;
   }
}
```

Appendix D.0 The main application file dfl.

```
#! /usr/local/bin/wish -f

#This is the main executable of the Design Flow Manager by Kris Kobylinski
#ECE 1997

##########################
#VIEW-MAIN GUI COMPONENTS#
##########################
set m .menu
wm title . "Design Flow Manager"
wm geometry . 650x500
frame .menu -relief raised -borderwidth 2 -background SeaGreen2
frame .top -relief sunken -borderwidth 1 -background #3a7
frame .bottom -relief raised -borderwidth 1

frame .top.left -relief raised
frame .top.right -relief raised

pack .menu -side top -fill both
pack .top -side top -expand 1 -fill both
pack .bottom -side bottom -fill x

pack .top.left .top.right -in .top -side left -expand 1 -fill both -padx 2m -pady 2m
#########################################################################

###############
#VIEW-MESSAGES#
#############################################################
#This messages are used for displaying in the box_ series and help boxes#
#############################################################
set msg1 "Design Flow Manager version 1.0 by Kris Kobylinski 1997 Department \ of Electrical and Computer Engineering University of Manitoba"
```

```
set msg2 "Do you really want to exit Design Flow Launcher ?"

set msg3 "Enter userid:"

set msg4 "Enter library name:"

set msg5 "Enter flow name:"

set msg6 "Enter flow location:"

set msg7 "Enter flow's icone location:"

set msg8 "Number of flow steps:"

set msg9 "Enter flow step names"

set msg10 "Choose the number of flow steps:"

set msg11 "To start search choose the steps from the \"Active Library Flow Step\" list by double mouse click. Next press the \"Start Searching\" button. If you want to add more steps after receiving searching result you may do so by clicking with the left mouse button on more steps. In case you want to run a new search with new set of steps, press the \"New Search\" button, to clear the list of required steps."

set msg12 "To view flow steps of a particular flow, double click with the left mouse button on the flow's name in the \"Active Library Flows\" list box."

set msg13 "You may activate any of the listed libraries by double clicking on the library name with the left mouse button."

set msg14 "You may launch any of the listed flows by double clicking on the flow's name with the left mouse button."

set msg15 "The step names can not include spaces, to separate words use underscores or start each word with an upper case character."

############################################################


############################
#CONTROLLER-GLOBAL VARIABLES#
############################

set exit 0

set userid ""

set path [pwd]

set entry {}

set user $env(USER)

set home $env(DFL_HOME)

set activelib eda_lib

set flowname {}

set flowlocation {}

set iconelocation {}
```

```
set stepsnumber {}
set location [ pwd ]
set flowstep(16) .
################################

#############################
#VIEW&CONTROLLER-PROCEDURES#
#############################

proc getFile {} {

  #this reads from the buffer file

  global entry
  global home
  set entry ""
  set str {}
  set buffer [ open "$home/buffer" r+ ]
  while { [ gets "$buffer" entry ] >= 0 } {
    set entry
    lappend str $entry
    }
  close $buffer
  return $str
}

proc getSteps {} {

  #this reads from the flowsteps array

  global stepsnumber
  global flowstep
  set str {}
```

```
    set i 0
    while { $i < $stepsnumber } {
      lappend str $flowstep($i)
      incr i +1
      }
    return $str
}

proc box_info { title bitmap text } {

    #This script creates information or warning box.
    #There are expected three arguments passed with the
    #invokation command:
    #->title of the box,
    #->internal bitmap name ( ex: error, hourglass, info,
    #  questhead, question, warning )
    #->message text.
    set win .boxinfo
    catch { destroy $win }
    toplevel $win
    wm title $win $title
    #wm geometry . 300x200
    frame $win.up -relief raised -borderwidth 1
    frame $win.down -relief raised -borderwidth 1
    label $win.bitmap -bitmap $bitmap
    message $win.msg -width 65m -justify center -text $text
    button $win.ok -text Ok -command "destroy $win"
    pack $win.up -expand 1 -fill both
    pack $win.down -expand 1 -fill both -ipadx 6m
    pack $win.bitmap $win.msg -in $win.up -side left -padx 3m -pady 4m
    pack $win.ok -in $win.down -expand 1 -fill x -padx 30m -pady 4m
}
```

```
proc list_box { title file msg } {

  #this creates the list box which returns a choice from the list
  #used by openFlow and listExistingLibraries
  #Command line arguments:
  #title - of the window
  #file - path and name to the file consisting the items to be read
  #msg - text displayed in the help window

  global home
  global activelib
  global entry
  set win .listbox
  catch { destroy $win }
  toplevel $win
  wm title $win $title

  frame $win.frame -relief raised -borderwidth 1
  frame $win.buttons -relief raised -borderwidth 1

  pack $win.frame -expand 1 -fill both
  pack $win.buttons -expand 1 -fill both

  listbox $win.items -relief raised -borderwidth 2 -yscrollcommand "$win.scroll set"
  scrollbar $win.scroll -relief sunken -borderwidth 2 -command "$win.items yview"

  pack $win.scroll -in $win.frame -side right -fill y
  pack $win.items -in $win.frame -side left -fill both

  button $win.ok -text OK -borderwidth 4 -command " destroy $win"
  button $win.help -text Help -borderwidth 4 -command "
    box_info \"$title Help\" info \"$msg\""
```

```
    pack $win.ok $win.help -in $win.buttons -side left -expand 1 -fill both -padx 2m -pady 2m

    set input [ open $file r ]
    while { [ gets $input line ] >= 0 } {
       $win.items insert end  [ string trimright "$line" \# ]
       }

    bind $win.items <Double-Button-1> {
      set entry [ selection get ]
      }

    tkwait variable entry
}

proc uni_box_enter { title number msg } {

    #this aquires same kind multi entry information like design steps
    #used by addFlow
    #Command line arguments:
    #title - of the window
    #number - of the entry fields
    #msg - text displayed in the help window

    global home
    global flowstep
    set win .uniboxenter
    catch { destroy $win }
    toplevel $win
    wm title $win $title

    frame $win.frame -relief raised -borderwidth 1
    frame $win.buttons -relief raised -borderwidth 1
```

```
    pack $win.frame -expand 1 -fill both
    pack $win.buttons -expand 1 -fill both

    set buffer [ open "$home/buffer" w+ ]
    puts $buffer ""
    close $buffer
    set buffer [ open "$home/buffer" a+ ]

    set i 0
  while { $i < $number  } {
    entry $win.stepentry$i -width 40 -relief sunken -bd 2 -textvariable flowstep($i)
    #lappend fsteps $flowstep($i)
    #puts $buffer $flowstep($i)
    incr i +1
    }

    close $buffer

    set i 0
    while { $i < $number } {
        pack $win.stepentry$i -in $win.frame -side top -expand 1 -fill x -padx 4m -pady 1m
        incr i +1
        }

    button $win.help -text Help -command "box_info \"$title Help\" info \"$msg\""

    button $win.ok -text Ok -command {
      destroy .uniboxenter
      }

     pack $win.ok $win.help -in $win.buttons -side left -expand 1 -fill x -padx 4m -pady 1
    tkwait window $win
  }
```

```
proc box_enter {title bitmap msg} {

  #this aquires one value and puts it in the entry
  #Used by newUser, delUser, newLib, delLib, delFlow, openLib
  #Command line arguments:
  #title - of the window
  #bitmap - displayed left to the entry field
  #msg - of the entry prompt

  global entry
  set entry {}
  set w .entrybox
  catch {destroy $w}
  toplevel $w
  wm title $w $title
  wm geometry $w 350x160

  frame $w.up -relief raised -borderwidth 1
  frame $w.down -relief raised -borderwidth 1
  label $w.bitmap -bitmap $bitmap
  message $w.msg -width 310 -justify center -text $msg
  entry $w.in -width 20 -relief sunken -bd 2 -textvariable entry

  button $w.cancel -text Cancel -command {
    set entry {}
    destroy .entrybox
     }
  button $w.ok -text Ok -command {
    destroy .entrybox
    }

  pack $w.up -expand 1 -fill both
  pack $w.down -expand 1 -fill both -ipadx 6m
```

```
    pack $w.bitmap $w.msg $w.in -in $w.up -side left -fill x -padx 4m -pady 5m
    pack $w.cancel $w.ok -in $w.down -side left -expand 1 -fill x -padx 4m -pady 1m

  tkwait window $w
}

proc multi_box_enter {title bitmap msg msg01 msg02 msg03 msg04 msg05 } {

  #this aquires flow information regarding its name, location and icon location
  #Used by: addFlow
  #Command line arguments:
  #title - of the window
  #bitmap, msg03 - redundand
  #msg, msg01, msg02 - entry field prompts
  #msg04 - enter steps button text
  #msg05 - number of steps prompt

  global entry
  global stepsnumber
  global msg15
  set entry {}
  set w .multientrybox
  catch {destroy $w}
  toplevel $w
  wm title $w $title

  frame $w.up -relief raised -borderwidth 1
  frame $w.up1 -relief raised -borderwidth 1
  frame $w.up2 -relief raised -borderwidth 1
  frame $w.up3 -relief raised -borderwidth 1
  frame $w.up4 -relief raised -borderwidth 1
  frame $w.down -relief raised -borderwidth 1
```

```
#FLOW NAME
message $w.msg -width 310 -justify center -text $msg
entry $w.in -width 30 -relief sunken -bd 2 -textvariable flowname

#FLOW LOCATION
message $w.msg1 -width 310 -justify center -text $msg01
entry $w.in1 -width 30 -relief sunken -bd 2 -textvariable flowlocation

#ICON LOCATION
message $w.msg2 -width 310 -justify center -text $msg02
entry $w.in2 -width 30 -relief sunken -bd 2 -textvariable iconelocation

#NUMBER OF FLOW STEPS
message $w.msg4 -width 310 -justify center -text $msg05
set j 15
while { $j > 0 } {
   radiobutton $w.rb$j -text $j -variable stepsnumber -value $j -anchor s
   incr j -1
   }

#BUTTONS
button $w.cancel -text Cancel -command {
   set entry {}
   set stepsnumber 0
   destroy .multientrybox
   }
button $w.ok -text Ok -command {
    destroy .multientrybox
    }
button $w.entersteps -text $msg04 -command {
    uni_box_enter "Enter flow step names" "$stepsnumber"  "$msg15"
}
```

```
    pack $w.up -expand 1 -fill both
    pack $w.up1 -expand 1 -fill both
    pack $w.up2 -expand 1 -fill both
    pack $w.up3 -expand 1 -fill both
    pack $w.down -expand 1 -fill both -ipadx 6m

    #FLOW NAME
    pack $w.msg  -in $w.up -side left -expand 1 -fill x -padx 4m -pady 1m
    pack $w.in -in $w.up -side right -expand 1 -fill x -padx 4m -pady 1m

    #FLOW LOCATION
    pack $w.msg1  -in $w.up1 -side left -expand 1 -fill x -padx 4m -pady 1m
    pack $w.in1 -in $w.up1 -side right -expand 1 -fill x -padx 4m -pady 1m

    #ICON LOCATION
    pack $w.msg2  -in $w.up2 -side left -expand 1 -fill x -padx 4m -pady 1m
    pack $w.in2 -in $w.up2 -side right -expand 1 -fill x -padx 4m -pady 1m

    #NUMBER OF FLOW STEPS
    pack $w.msg4 -in $w.up3 -side top -fill x
    set j 1
    while { $j <= 15 } {
        pack $w.rb$j -in $w.up3 -side left  -fill x -padx 0m -pady 1m
        incr j +1
        }

    pack $w.entersteps $w.cancel $w.ok -in $w.down -side left -expand 1 -fill x -padx 4m -pady 1m

    tkwait window $w
}
```

```
proc step_box { } {

  #this display list of flows one list box and steps of chosen flow in another
  #Used by: viewSteps

  global activelib
  global home
  set sb .win
  toplevel $sb
  wm title $sb "Step Viewer"

  frame $sb.top -relief sunken -borderwidth 2
  frame $sb.buttons -relief sunken -borderwidth 2
  pack $sb.top $sb.buttons -side top

  frame $sb.left -relief sunken -borderwidth 1
  frame $sb.right -relief sunken -borderwidth 1
  pack $sb.left $sb.right -in $sb.top  -side left -expand 1 -fill both -padx 2m -pady 2m

  label $sb.label_flows -text "Active Library Flows" -relief raised -borderwidth 2
  scrollbar $sb.scroll_flows -relief sunken -borderwidth 2 -command "$sb.flows yview"
  listbox $sb.flows -relief raised -borderwidth 2 -yscrollcommand "$sb.scroll_flows set"

  label $sb.label_steps -text "Flow Steps" -relief raised -borderwidth 2
  scrollbar $sb.scroll_steps -relief sunken -borderwidth 2 -command "$sb.steps yview"
  listbox $sb.steps -relief raised -borderwidth 2 -yscrollcommand "$sb.scroll_steps set"

  button $sb.ok -text OK -borderwidth 4 -command " destroy $sb"
  button $sb.help -text Help -borderwidth 4 -command {
     box_info "Step Viewer Help" info $msg12
  }

  pack $sb.label_flows -in $sb.left -side top -fill x
```

```
    pack $sb.scroll_flows -in $sb.left -side right -fill y
    pack $sb.flows -in $sb.left -side left -fill both

    pack $sb.label_steps -in $sb.right -side top -fill x
    pack $sb.scroll_steps -in $sb.right -side right -fill y
    pack $sb.steps -in $sb.right -side left -fill both

    pack $sb.ok $sb.help -in $sb.buttons -side left -expand 1 -fill both -padx 2m -pady 2m

    #FILLING THE LIST BOXES
    set f [ open "$home/$activelib/flow_list.names" ]
    while { [ gets $f line ] >= 0 } {
       $sb.flows insert end  [ string trimright "$line" \# ]
       }
    close $f

    bind $sb.flows <Double-Button-1> {

       .win.steps delete 0 end
       set f [ open "$home/$activelib/[ selection get ]/flow_step_list.names" ]
       while { [ gets $f line ] >= 0 } {
            .win.steps insert end  [ string trimright "$line" \# ]
            }
       close $f
    }
}

proc search_box_enter { } {

   #this displays all flow steps and chosen flow steps, runs the search program and displays resulting statistics
   #Used by: searchFlow

   global activelib
   global home
```

```
set sbe .win
toplevel $sbe
wm title $sbe "Flow Searcher"

frame $sbe.top -relief sunken -borderwidth 1
frame $sbe.bottom -relief sunken -borderwidth 2
frame $sbe.message -relief sunken -borderwidth 1

frame $sbe.top.left -relief sunken
frame $sbe.top.centre -relief sunken
frame $sbe.top.right -relief sunken

pack $sbe.top -side top -fill both
pack $sbe.bottom -side top -expand 1 -fill both
pack $sbe.message -side bottom -expand 1 -fill both

pack $sbe.top.left $sbe.top.centre $sbe.top.right -in $sbe.top -side left -expand 1 -fill both -padx 2m -pady 2m

# LEFT FRAME

label $sbe.label_keywords -text "Active Library Flow Steps" -relief raised -borderwidth 2
scrollbar $sbe.scroll_keywords -relief sunken -borderwidth 2 -command "$sbe.keywords yview"
listbox $sbe.keywords -relief raised -borderwidth 2 -yscrollcommand "$sbe.scroll_keywords set"

# RIGHT FRAME

label $sbe.label_rules -text "Required Flow Steps" -relief raised -borderwidth 2
listbox $sbe.rules -relief raised -borderwidth 2 -yscrollcommand "$sbe.scroll_rules set"
scrollbar $sbe.scroll_rules -relief sunken -borderwidth 2 -command "$sbe.rules yview"

# BUTTONS

button $sbe.ok -text OK -borderwidth 4 -command "destroy $sbe"
```

```
    button $sbe.new -text "New Search" -borderwidth 4 -command {
      set buffer [ open buffer w+ ]
      puts $buffer ""
     #global sbe
     .win.rules delete 0 end
      }

    button $sbe.search -text "Start Searching" -borderwidth 4 -command {
      set buffer1 [ open buffer1 w+ ]
      puts $buffer1 ""
      set buffer1 [ open buffer1 a+ ]
      set steps [ getFile ]
      eval exec $home/search $activelib << "space $steps"  >buffer1
      while { [ gets "$buffer1" llline ] >= 0 } {
      .win.result insert end $llline
      }
  close $buffer1
  }

    button $sbe.help -text Help -borderwidth 4 -command {
      box_info "Search Help" info $msg11
    }

 # MESSAGE BOX

  listbox $sbe.result -relief sunken -borderwidth 2 -yscrollcommand "$sbe.scroll_info set"
  scrollbar $sbe.scroll_info -relief sunken -borderwidth 2 -command "$sbe.result yview"
  label $sbe.status -text "Search Result"  -relief raised -borderwidth 1

 # PACKING

  pack $sbe.label_keywords -in  $sbe.top.left -side top -fill x
```

```
pack $sbe.scroll_keywords -in $sbe.top.left -side right -fill y
pack $sbe.keywords -in $sbe.top.left -side left -expand 1 -fill both

pack $sbe.label_rules -in $sbe.top.right -side top -fill x
pack $sbe.scroll_rules -in $sbe.top.right -side right -fill y
pack $sbe.rules -in $sbe.top.right -side left -expand 1 -fill both

pack $sbe.ok $sbe.new $sbe.search $sbe.help -in $sbe.bottom -side left -expand 1 -fill both -padx 8m -pady 4m

pack $sbe.status -in $sbe.message -side top -expand 1 -fill x
pack $sbe.result -in $sbe.message -side left -expand 1 -fill x
pack $sbe.scroll_info -in $sbe.message -side right -fill y

# FILLING LIST BOXES

set f [ open "$home/$activelib/flow_list.names" ]
while { [ gets $f line ] >= 0 } {
   set file [ open "$home/$activelib/[ string trimright "$line" # ]/flow_step_list.names" ]
   while { [ gets $file lline ] >= 0 } {
      $sbe.keywords insert end  [ string trimright "$lline" \# ]
      }}
close $f
close $file

set buffer [ open buffer w+ ]
puts $buffer ""
.win.rules delete 0 end
bind $sbe.keywords <Double-Button-1> {
   global sbe
   set buffer [ open buffer a+ ]
   puts $buffer [ selection get ]
   .win.rules insert end [ selection get ]
   close $buffer
```

```
    }
}

proc starTool {tool_name tool} {
    .info_field insert end "Starting $tool_name ..."
    update
    exec $tool
    }

#####################################
#CONTROLLER-MENU OPTION PROCEDURES#
#####################################

proc viewSteps { } {
   step_box
}

proc openFlow {} {
   global entry
   global home
   global activelib
   global msg14
    list_box  "Open Flow" "$home/$activelib/flow_list.names" "$msg14"
    if { $entry != "" } {
        .info_field insert end "$entry is being executed"
        exec fx -open /home/ee/u15/kobylin/flow_lib/$entry -multiuser 10
        }
}

proc runFlow {flow_name} {
    set flow_name pcb_mentor_quantic
    flow_icons itemconfigure $flow_name -foreground red
    update
```

```
    .info_field insert end "Starting $flow_name flow..."
    update
    exec fx -open /home/ee/u15/kobylin/flow_lib/$flow_name.flow -multiuser 10
    focus .flow_icons
    .flow_icons itemconfigure $flow_name -foreground blue
    update
}

proc searchFlow { } {
   search_box_enter
}

proc getValue {offset maxBytes} {
    global state
    set last [expr $offset+$maxBytes-1]
    string range $state $offset $last
    }

proc selGone {} {
    global state
    set state {}
    }

proc libList { } {
   global msg13
   global home
   global activelib
   global entry
   list_box  "List Existing Libraries" "$home/lib.names" "$msg13"
   if { $entry!= "" } {
      set activelib $entry
   }
}
```

```
proc addadm msg {
   global entry
   global user
   global home
   set entry { }
   box_enter "Grant User Administration Priviledges" hourglass $msg
   if { $entry!= "" } {
        exec $home/addadmin $user $entry
        .info_field insert end "$entry is being added to the admin list."
        }
}

proc deladm msg {
   global entry
   global user
   global home
   set entry { }
   box_enter "Revoke User Administration Priviledges" hourglass $msg
   if { $entry != "" } {
      exec $home/deladmin $user $entry
      .info_field insert end "$entry is being deleted from the admin list."
      }
}

proc newlibrary msg {
   global entry
   global user
   global home
   box_enter "Create new flow library" hourglass $msg
   if { $entry != "" } {
        exec mkdir $entry
        exec $home/newlib $user $entry
```

```
        .info_field insert end "$entry flow library is being created."
    }
}

proc dellibrary msg {
    global entry
    global user
    global home
    box_enter "Delete flow library" hourglass $msg
    if { $entry != "" } {
        exec $home/dellib $user $entry
        .info_field insert end "$entry flow library is being deleted."
    }
}

proc addflow { msg msg01 msg02 msg03 msg04 msg05 } {
    global user
    global home
    global activelib
    global flowname
    global flowlocation
    global iconelocation
    global flowstep
    global fsteps
    global stepsnumber
    multi_box_enter "Add flow to the active library" hourglass $msg $msg01 $msg02 $msg03 $msg04 $msg05
    if { $flowname != "" } {
        .info_field insert end [ getFile ]
        if { $activelib != "" } {
            exec mkdir "$activelib/$flowname"
            set steps [ getSteps ]
            eval exec $home/newflow $user $activelib $flowname $flowlocation $iconelocation << "space $steps"
            .info_field insert end "$flowname has been added."
```

```
        }
    }
}

proc deleteflow msg {
    global entry
    global user
    global home
    global activelib
    box_enter "Delete flow from the active library" hourglass $msg
    if { $entry != "" } {
      exec $home/delflow $user $activelib $entry
      .info_field insert end "$entry flow is being deleted."
      }
}

proc openlibrary msg {
    global activelib
    global entry
    box_enter "Set active library" hourglass $msg
    if { $entry != "" } {
        set activelib $entry
        .info_field insert end "$activelib library is being activated."
    }
}

####################
#VIEW - GUI DEFINITION#
########################
#box_ series are procedures.#
########################

menubutton .menu.flow -text Flow -menu .menu.flow.m -underline 0
```

```
menu .menu.flow.m
.menu.flow.m add command -label New -underline 0 -command { starTool XpertBuilder xb }
.menu.flow.m add command -label "View Design Steps" -underline 0 -command { viewSteps }
.menu.flow.m add command -label Open -underline 0 -command { openFlow }
.menu.flow.m add command -label Search -underline 0 -command { searchFlow }
.menu.flow.m add separator
.menu.flow.m add command -label Exit -underline 0 -command { destroy .}

menubutton .menu.library -text Library -menu .menu.library.m -underline 0

menu .menu.library.m
.menu.library.m add command -label "List Existing Libaries" -underline 0 -command { libList }
.menu.library.m add command -label "Open Library" -underline 0 -command { openlibrary $msg4 }

menubutton .menu.admin -text Administer -menu .menu.admin.m -underline 0

menu .menu.admin.m
.menu.admin.m add command -label "Grant User Administration Privileges" -underline 0 -command {  addadm
$msg3 }
.menu.admin.m add command -label "Revoke User Administration Privileges" -underline 0 -command { deladm
$msg3 }
.menu.admin.m add separator
.menu.admin.m add command -label "Create New Library" -underline 0 -command { newlibrary $msg4 }
.menu.admin.m add command -label "Delete Library" -underline 0 -command { dellibrary $msg4 }
.menu.admin.m add separator
.menu.admin.m add command -label "Add Flow" -underline 0 -command { addflow $msg5 $msg6 $msg7 $msg8
$msg9 $msg10 }
.menu.admin.m add command -label "Delete Flow" -underline 0 -command { deleteflow $msg5 }

menubutton .menu.help -text Help -menu .menu.help.m -underline 0

menu .menu.help.m
.menu.help.m add command -label Contents
```

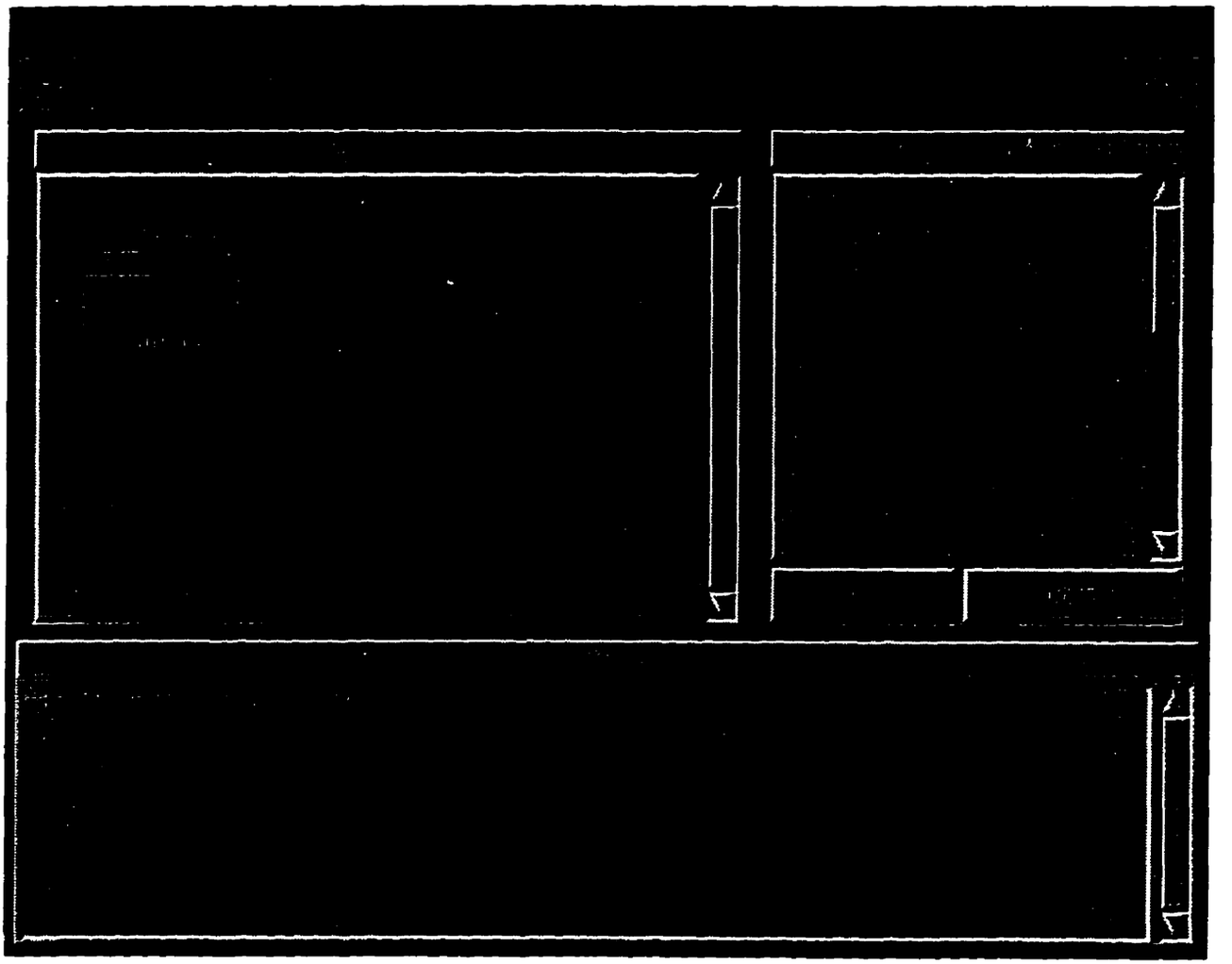
```
.menu.help.m add separator
.menu.help.m add command -label About... -underline 0 -command { box_info "About Design Flow Manager" info $msg1 }

# LEFT FRAME

label .label_flowmanager -text "Active library $activelib" -relief raised -borderwidth 2
update idletasks
.label_flowmanager config -text "Active library $activelib"
canvas .flow_icons -relief raised -borderwidth 2 -yscrollcommand ".scroll_flowmanager set"
scrollbar .scroll_flowmanager -relief sunken -borderwidth 2 -command ".flow_icons yview"

set flist [ open "$home/$activelib/flow_list.names" ]
while { [ gets $flist line ] >= 0 } {
   set iloc [ open "$home/$activelib/[ string trimright "$line" "\#" ]/icone.dfl" ]
   gets $iloc iline
   eval .flow_icons create bitmap 2c 2c -bitmap @$iline -tags activeicon
   close $iloc
   set floc [ open "$home/$activelib/[ string trimright "$line" "\#" ]/location.dfl" ]
   gets $floc fline
   .flow_icons bind activeicon <Double-Button-1> { runFlow $fline }
   close $floc
}

# RIGHT FRAME

label .label_navigator -text $location -relief raised -borderwidth 2
update
```

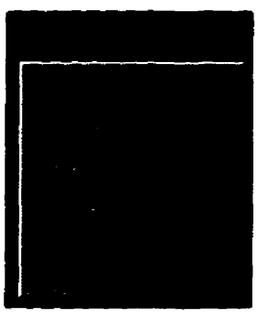
```
listbox .file_names -relief raised -borderwidth 2 -yscrollcommand ".scroll_navigator set"
scrollbar .scroll_navigator -relief sunken -borderwidth 2 -command ".file_names yview"

foreach i [lsort [glob -nocomplain *]] {
     .file_names insert end $i
}

frame .buttons
button .up_file -text Up -command {
   .file_names delete 0 end
   set path [file dirname [pwd]]
   set location $path
   cd $path
   .info_field insert end $path
   foreach i [lsort [glob -nocomplain *]] {
   .file_names insert end $i
}}

button .down_file -text Down -command {
#   .file_names delete 0 end
#   selection handle .down_file getValue STRING
   set pathdown [selection get]
   set location $pathdown
   .file_names delete 0 end
#   .down_file config -command {selection own .down_file selGone}
#   .info_field insert end $pathdown
   cd $pathdown
   .info_field insert end $pathdown
   foreach i [lsort [glob -nocomplain *]] {
   .file_names insert end $i
}}

# BOTTOM
```

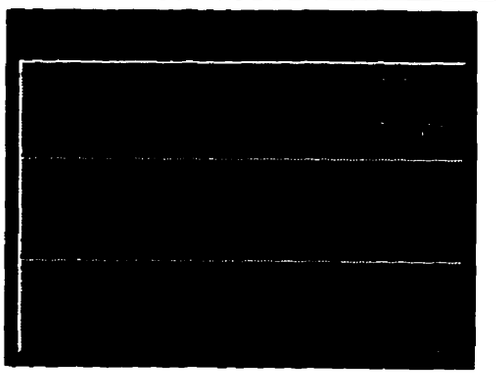
```
listbox .info_field -relief sunken -borderwidth 2 -yscrollcommand ".scroll_info set"
scrollbar .scroll_info -relief sunken -borderwidth 2 -command ".info_field yview"
label .status -text Status -relief raised -borderwidth 1

# PACKING

pack .menu.flow .menu.library .menu.admin -side left
pack .menu.help -side right

pack .label_flowmanager -in .top.left -side top -fill x
pack .scroll_flowmanager -in .top.left -side right -fill y
pack .flow_icons -in .top.left -side left -expand 1 -fill both

pack .label_navigator -in .top.right -side top -fill x
pack .buttons -in .top.right -side bottom -fill x
pack .scroll_navigator -in .top.right -side right -fill y
pack .file_names -in .top.right -side left -expand 1 -fill both
pack .up_file .down_file -in .buttons -side left -expand 1 -fill x

pack .status -in .bottom -side top -expand 1 -fill x
pack .info_field -in .bottom -side left -expand 1 -fill x
pack .scroll_info -in .bottom -side right -fill y

update

#this is the end
```

Appendix E.0 The final GUI version.

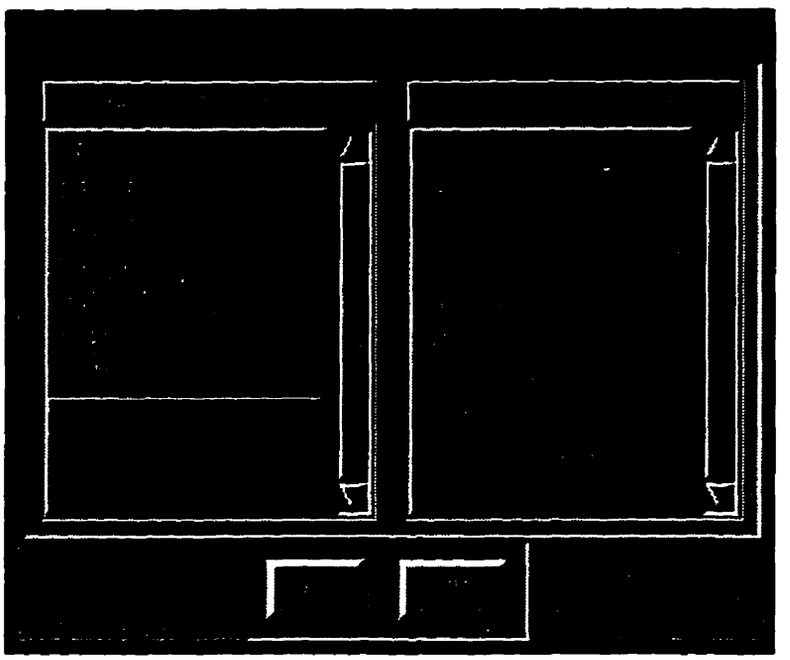
E.1 Main window

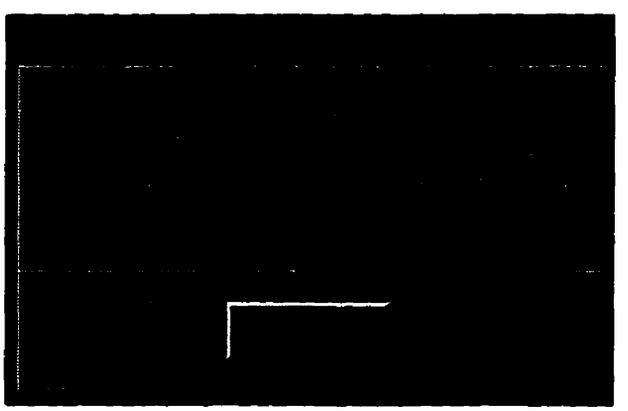




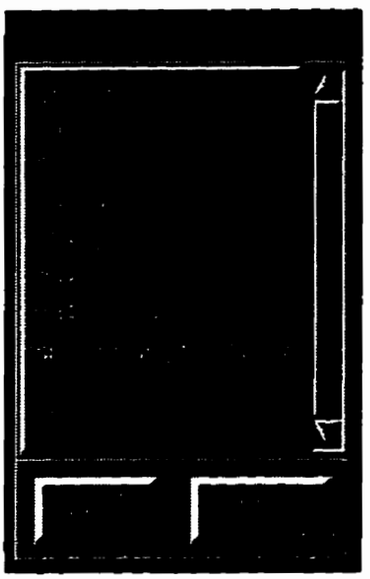


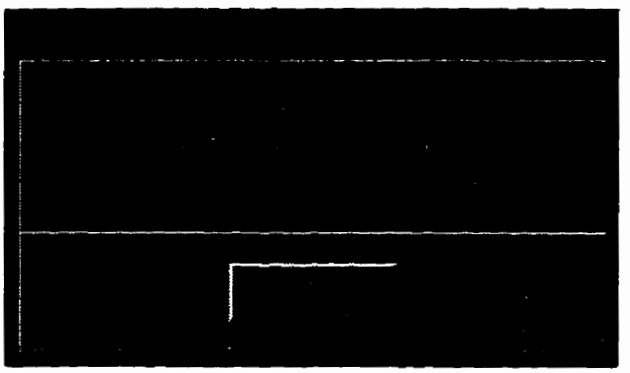
E.2 ViewDesignSteps



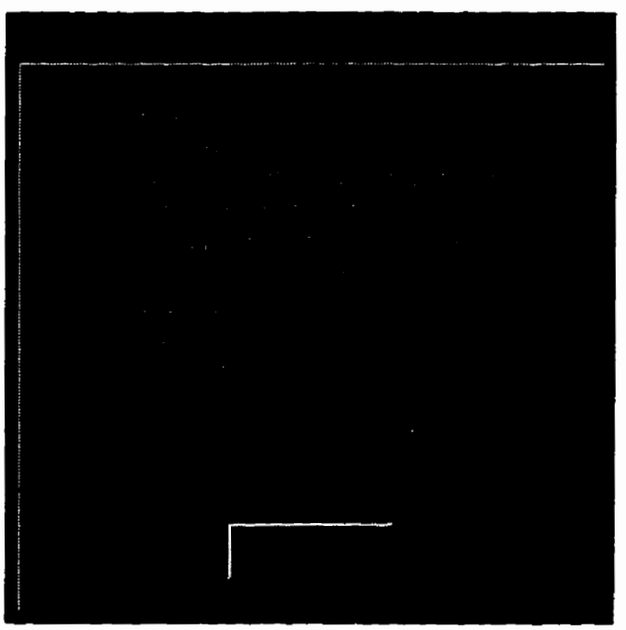


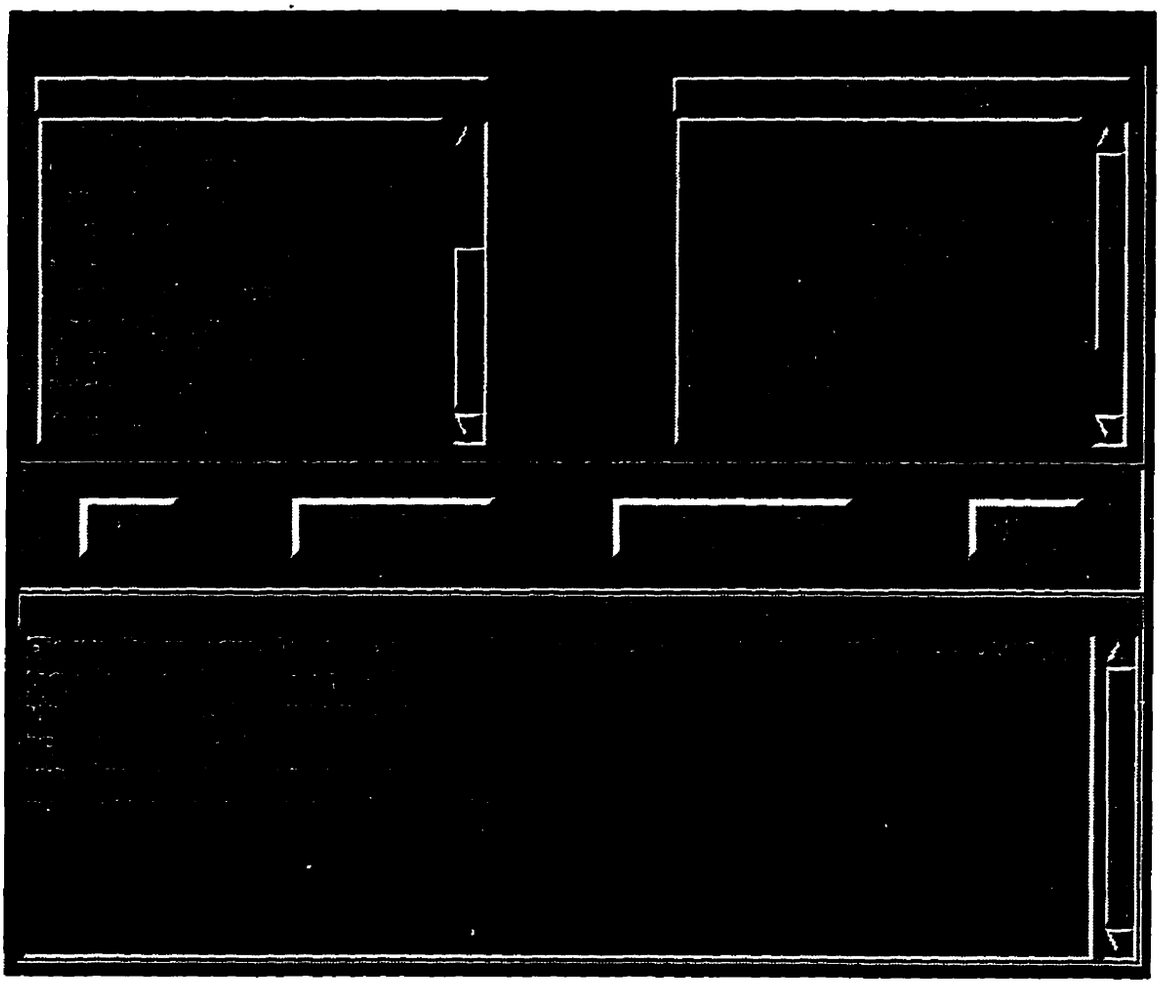
E.3 OpenFlow



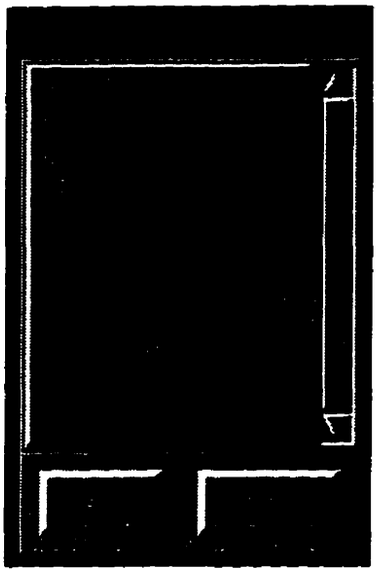


E.4 SearchFlow

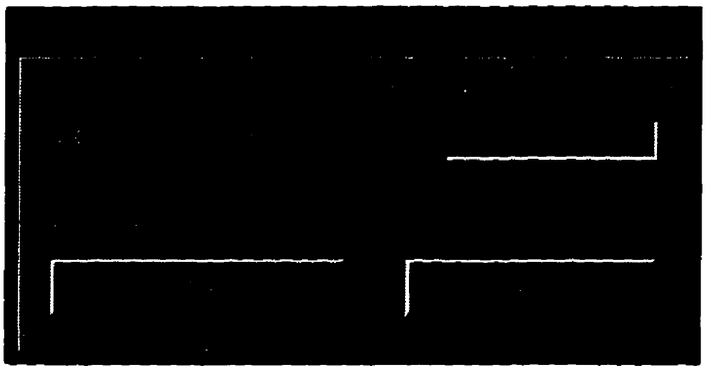




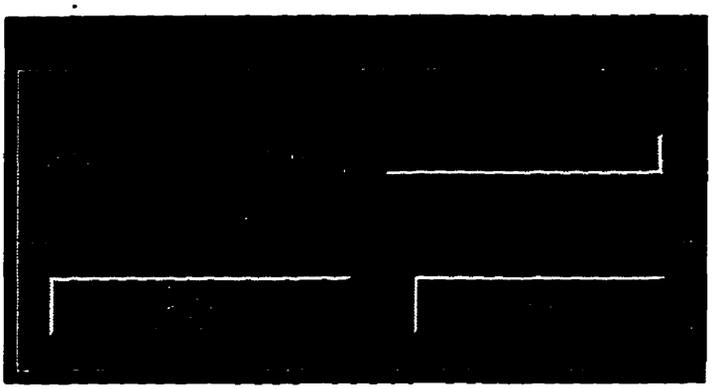
E.5 ListExistingLibraries



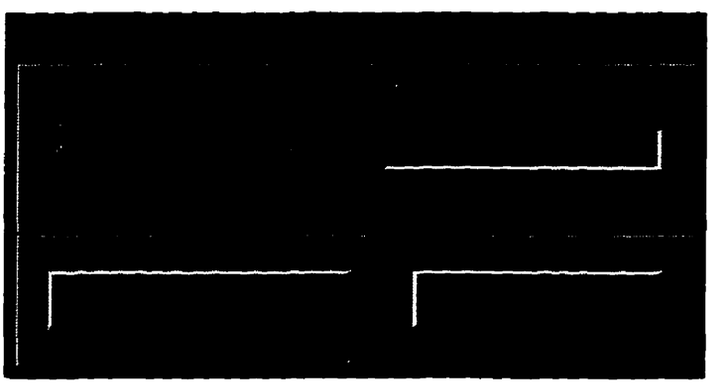
E.6 OpenLib



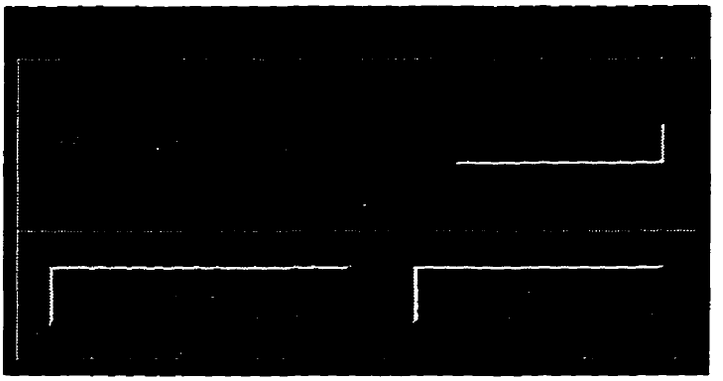
E.7 NewAdmin



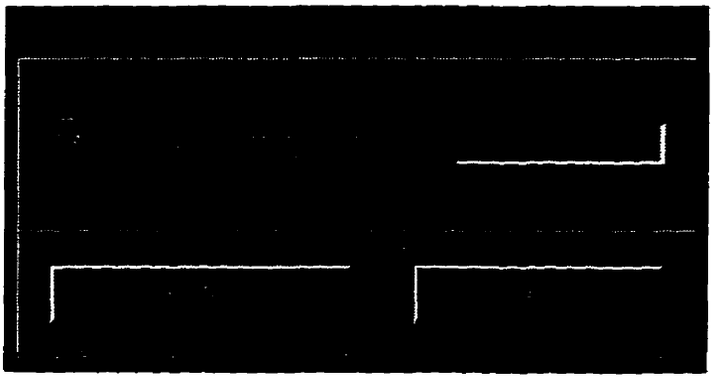
E.8 DelAdmin



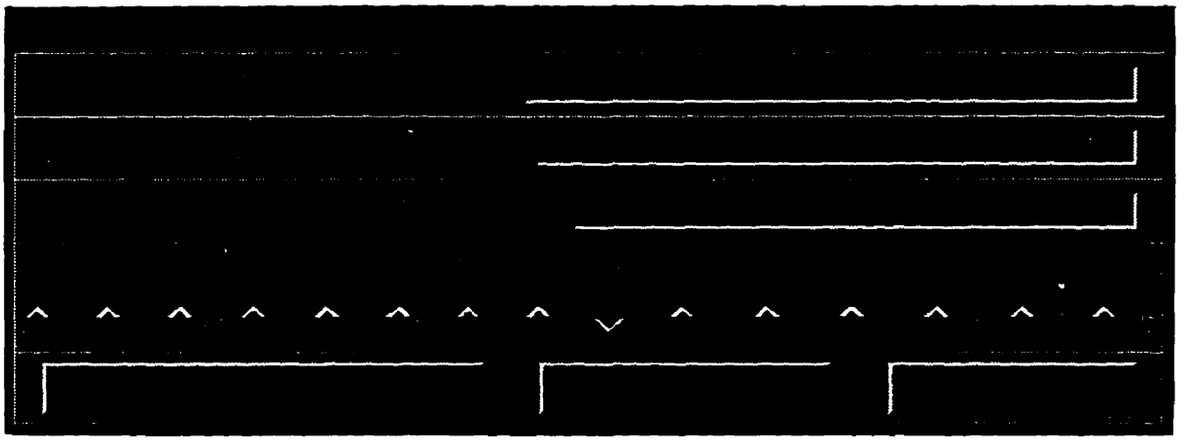
E.9 NewLib

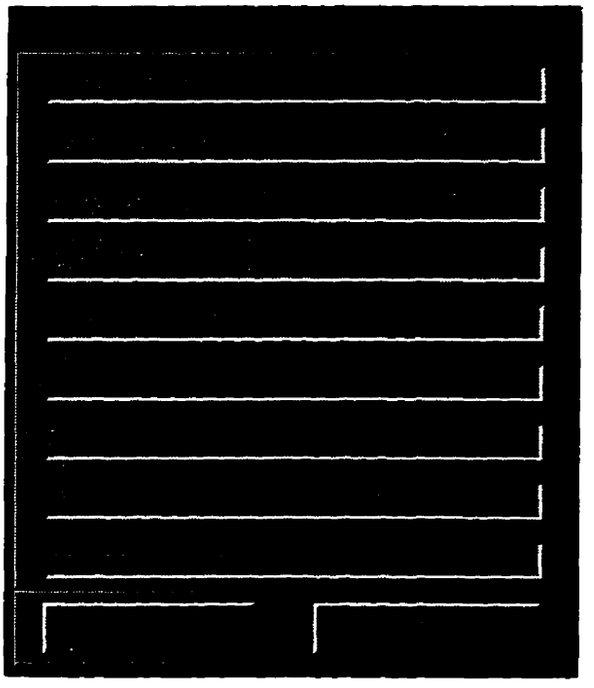


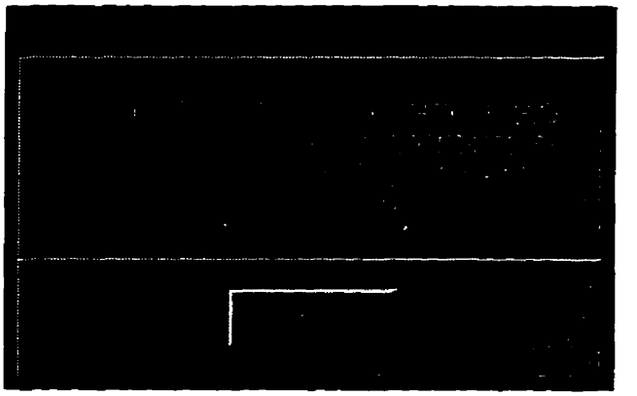
E.10 DelLib



E.11 AddFlow







E.12 DelFlow