University of Alberta

Reverse Engineering Legacy User Interfaces Using Interaction Traces

By

©

Mohammad El-Ramly

A thesis submitted to the Faculty of Graduate Studies and Research in partial fulfillment of the requirements for the degree of Doctor of Philosophy

Department of Computing Science

Edmonton, Alberta

Fall 2003

Your file Votre référence
ISBN: 0-612-87965-8
Our file Notre référence
ISBN: 0-612-87965-8

University of Alberta

Library Release Form

Name of Author: Mohammad Mahmoud Fawzi El-Ramly

Title of Thesis: Reverse Engineering Legacy User Interfaces Using Interaction Traces

Degree: Doctor of Philosophy

Year This Degree Granted: 2003

Date: 10th June, 2003

University of Alberta

Faculty of Graduate Studies and Research

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research for acceptance, a thesis entitled ***Reverse Engineering Legacy User Interfaces Using Interaction Traces*** submitted by **Mohammad El-Ramly** in partial fulfillment of the requirements for the degree of **Doctor of Philosophy**.

[illegible]

Eleni Stroulia, Associate Professor
(Supervisor)

[illegible]

Paul Sorenson, Professor
(Co-supervisor)

[illegible]

Witold Pedrycz, Professor

Osmar R. Zaïane, Assistant Professor

[illegible]

Gail Murphy, Associate Professor
(External Examiner)

Date: May 20th, 2003

*To the most precious ones
who gave me life and their
unconditional, unlimited love
and support,*

My Parents,

*To my sunshine, my only
sunshine, who makes me
happy. To my precious
sweetheart,*

My Wife Aisha,

*To the little one who can
make me smile even during
the hardest times,*

My Son Mahmoud,

*To my brother, friend
and supporter,*

Hassan,

***With all my love,
I dedicate this work.***

ABSTRACT

Legacy system user-interface reengineering is an increasingly popular area in research and practice. Many legacy user-interfaces get reengineered to reproduce them in modern graphical user-interfaces, integrate them with other systems' front-ends, or most important, open them for Web-access. Often, it is desired to reengineer the user-interface without changing the legacy system code because the system performance is satisfactory and/or due to the prohibitive cost or risk. In such cases, lightweight non-invasive reengineering methods are needed.

This thesis presents a novel method for reverse engineering legacy character-based user-interfaces using traces of interaction between the legacy system and its users, as the only input. This "interaction reverse engineering" method produces a behavioral model of the legacy user-interface and discovers important usage scenarios of the legacy system services, represented by the frequent patterns of interaction with its user-interface. Then, a complementary forward engineering method uses the model and patterns to build a new task-centered front-end.

Our method consists of three steps and is implemented in a prototype tool called the Legacy Navigation Domain Identifier (LeNDI). First, the system-user dialog is recorded in the form of interaction traces using a specially instrumented emulator. These traces capture the screen snapshots forwarded to the user terminal and the user keyboard actions in return. Second, LeNDI builds a behavioral state-transition model for the legacy user-interface, whose states represent the legacy user-interface screens and whose transitions represent the permissible user actions on each screen. To build the model, LeNDI extracts a vector of features for every snapshot, clusters similar snapshots together, and finally

induces a classifier that can classify new snapshots to one of the existing clusters. Third, LeNDI uses one of its two novel interaction pattern mining algorithms, IPM and IPM2, to mine the interaction traces for patterns of user activity. Associated with these steps, is a process of user feedback and revision to verify the results.

Our interaction reverse engineering method is code-independent and utilizes a novel easy-to-collect input, the interaction traces. Currently, it can reverse engineer block-mode data transfer protocols, e.g., IBM 3270. It is lightweight in terms of the time, cost and skills required. It supersedes the current manual labor-intensive time-consuming industrial practices. Several case studies were conducting to reverse engineer the user interaction with a number of real legacy systems, with very encouraging results.

ACKNOWLEDGMENT

Thanks are due to many people who, over the course of this study, provided me with enormous help and encouragement. This acknowledgment is but a small appreciation for their priceless support.

The guidance and vision of Professor Eleni Stroulia, my main supervisor, were instrumental in achieving the goals of this work. I thank her for her enthusiastic support, the unique mentoring environment she provided and the invaluable discussions we had. I also thank Professor Paul Sorenson, my co-supervisor, for his help and advice. His ideas, comments and advice are greatly appreciated.

The CelLEST project team at University of Alberta provided an exceptionally cooperative and friendly environment. In particular, thanks are due to Roland Penner for his great help in implementing the prototype tool, LeNDI, and Paul Iglinski for his work on top-down clustering and decision tree classifier induction algorithms. Thanks are due to Bruce Matichuk from Celcorp for the many fruitful discussions we had in the course of CelLEST project. I like to thank the summer students Brice Riemenschneider and Warren Blanchet for their help in implementing LeNDI.

I would like to acknowledge the generous support of NSERC and Celcorp to this research via an NSERC Industrial Postgraduate Scholarship 216077-98. I would like to thank Celcorp for giving me the opportunity to work on their site for a few months at the early stages of CelLEST project, for the resources they have allocated to me and for the wonderful support of their staff. That period of time was vital in familiarizing me with the state of the art user interface reengineering technology and for the progress of this research.

Table of Contents

List of Tables

List of Figures

Chapter One
Introduction

"By the time you will finish your thesis, the systems, tools and/or prototypes that you have developed will be legacy systems and you will need to reverse engineer them in order to understand, migrate and/or reengineer them."

Anonymous

1.1 Background

Over years of development and investment, business software systems, such as bank finance systems, customer relationship management (CRM) systems and airline reservation systems, grew in size and value. They contain the specifications for diverse business policies and corporate decisions and constitute some of the most important industrial assets for many companies [LBS94]. Corporations have invested substantially in developing these mainframe-based systems. They have almost spent as much to develop integrated reliable database systems. In the recent past, they invested even more money in making their systems Y2K and Euro compliant. [Sne00]

Many such systems were developed using the technology of the 1970s to mid-1980s. They have been modified many times by different programmers. As a result, they have become very complex and difficult to understand, maintain, renovate and/or reengineer [LBS94]. Such systems are referred to in the literature as "legacy systems". The Free Online Dictionary of Computing [FOLD96] defines a legacy system as:

Definition 1.1

"A computer system or application program which continues to be used because of the prohibitive cost of replacing or redesigning it and despite its poor competitiveness and compatibility with modern equivalents. The implication is that the system is large, monolithic and difficult to modify."

This definition includes hardware and software systems and is neither restricted to mainframe-based systems nor to a specific programming language or platform. Gold [Gold98] expands the definition of a legacy system to include not just the hardware and software but also the environment, the people, the procedures, etc., surrounding the system. He states:

Definition 1.2

"A legacy system is a socio-technical system containing legacy software"

This is because when a critical software system ages, not only understanding and modifying the system becomes hard, but also changing and modifying the surrounding environment, especially the people, becomes hard too. Bergey *et al.* [BSTWW99] consider resistance to change and the growth of a culture dependent on maintaining the status quo plus inadequate training programs as the third reason for failure of reengineering projects. This thesis adopts the first definition, but it is only concerned with software systems not hardware.

In return for the effort and investment spent, mainframe-based legacy systems have demonstrated robustness, reliability and scalability in providing business-critical processing needs. This is especially true where the application concerned involves huge numbers of transactions and many simultaneous users, as is the case with banking or airline reservation systems. Most important, however, is that many of the business processes of companies are encapsulated in the logic of legacy applications; they are, in effect, the repositories of hard-won corporate experience and knowledge, that may not be available in other formats [Att00]. Considering this, legacy systems will remain the Information Technology (IT) backbone for many corporations, for many years to come.

However, due to their age, many legacy systems suffer from some or all of the following aging disadvantages [Par94]:

1. Performance and functionality degradation.
2. Lack of coding standards, proper documentation and version control.
3. Incremental and patch updates to the code and design that often violate the original software design concepts. These "ignorant surgeries", as Parnas [Par94] calls them, result in degradation of the maintainability and comprehensibility of the software.

4. Pollution [Vis01], which is the accumulation of duplicate and dead code (that is either never compiled or never executed), useless components (e.g., reports that no user needs anymore), and dead data.
5. A general lack of understanding of the internal workings of the system and how its functions relate to its modules and data.
6. Significant resistance to modification and evolution, not only due to technical difficulties but also due to socio-political factors.
7. Character-based user interfaces (CUIs), which are not competitive with the superior alternatives offered by today's technology.
8. Great difficulty in integrating with other systems and the World-Wide-Web (WWW).

Because of these symptoms, many organizations are migrating, renovating or reengineering their legacy systems to achieve one or more of the following objectives:

1. Migrating the whole application to a newer, faster, and non-proprietary platform.
2. Enhancing system comprehensibility and maintainability, i.e., putting it under control.
3. Adding new substantial or minor functionality.
4. Integrating the legacy application with other applications, on legacy or new platforms.
5. Migrating the application user interface (UI) to a new platform.
6. Enabling access to the system through the WWW.

Depending on the goal of the reengineering effort and the current status of the legacy system, reengineering activities can vary widely. They range from rebuilding the system and using the legacy system as an input for the analysis and design phase, to wrapping the legacy system to fit it in a new computing environment, e.g., graphical user interfaces (GUIs), CORBA, client/server architectures, or the WWW.

1.2 Motivation

When the objective of a legacy system reengineering effort is migrating its UI to a new platform, enabling access to the system through the Web (Web-enabling) or lightweight front-end integration with other systems, then the reengineering effort can focus on enhancing the UI of the legacy system or developing a new UI. This is especially applicable, when the prime aging symptoms of the legacy systems are the last two of the eight mentioned above, or in other words, the legacy system is under control and exhibits satisfactory performance but its main weakness is its poor UI and its inability to be integrated with other systems. This occurs due to the legacy UI falling short in three areas: user access, usability and navigation at different levels [BB01]:

1. **User Access.** Most legacy systems are proprietary monolithic systems that were not designed with integration with the WWW or other technologies in mind. Usually they do not have clear separation between their presentation, logic and data layers, which makes opening a legacy system for access via a new platform or for front-end integration with other systems a hard task. The presentation layer refers to the source code that controls the UI. The logic layer refers to the application code that provides the main functionality. The data layer refers to the container of the application data and the code used to access it.
2. **Usability.** The old-looking "dumb" terminals, e.g., IBM 3270 and VT series, were quite adequate for their time in spite of being quite limited in their display capabilities. Legacy character-based UIs are non-intuitive and hard to learn. Their UIs dissatisfy today's users, who are used to graphical user interfaces and Web interfaces. Additionally, the learning curve of new users is slow and the training costs are high.
3. **Navigation.** Due to their limited presentation capabilities, legacy character-based UIs offer tedious navigation patterns to accomplish user tasks. For example, flipping a multi-page report may require using function keys or issuing some commands to move forward and backward between the many screens containing the report. Instead, in a GUI environment, a scroll bar enables instant access to any page of the report with a mouse click.

So, in many cases it is unnecessary, hard, expensive, risky and/or impossible to change the code of a legacy system and design, yet, it is desired to reengineer its UI. The goal of this reengineering is to open the system to the Web, Wireless Access Protocol (WAP) or other platforms, to integrate its front-end with those of other systems and/or to slightly extend its functionality. For these cases, there is a need for lightweight non-invasive UI reengineering methods. These methods need to be lightweight in the sense that they are cost-effective, semi-automated and relatively easy to deploy because they require moderate skills and low technology. And they are non-invasive in the sense that they almost do not need alteration of the legacy system. Industry has offered some primitive labor-intensive solutions to this problem.

The CelLEST project for UI reengineering [SES02, SEIS03] is a collaborative project between the Software Engineering Research Lab. at University of Alberta and an industrial partner; Celcorp [Cel]. The goal of CelLEST was to develop an intelligent semi-automated lightweight non-invasive method and prototype tools for legacy system CUI reengineering, Web-enabling and front-end integration. The method developed in this project takes as input recorded traces of interaction between the legacy system and its users through the legacy CUI (interaction traces) and does not require modifications to the legacy code. We call this approach to UI reengineering "interaction reengineering" as opposed to "code reengineering". The CelLEST interaction-reengineering method consists of two phases: a reverse engineering phase and a forward engineering phase. The focus of this thesis is the reverse engineering phase. It describes the novel legacy CUI reverse engineering method developed, during and after the CelLEST project. Since this method takes interaction traces as its only input, in effect, it adopts an "interaction reverse engineering" approach as opposed to "code reverse engineering".

1.3 The CelLEST Project

To achieve its goal of developing an intelligent semi-automated lightweight non-invasive method for CUI reengineering, CelLEST employs a mixture of document analysis, clustering, example-based modeling of user actions, visualization, data mining, task model inference, XML wrapping and automated GUI layout. The outcome is a novel CUI reengineering method that utilizes interaction traces and does not change the legacy system code or structure. The CelLEST method, resulting from the project, supersedes the current manual, labor intensive practices of legacy character-based UI modeling and reengineering, which need intensive human input, intuition and experience.

When invasive solutions are unnecessary, undesirable, too hard, impossible, risky and/or cost-ineffective and code and architecture modification is not mandatory for front-end reengineering of a legacy system, the CelLEST method is a very suitable solution. Since it is important to understand CelLEST method to understand the context of this thesis, the method is introduced briefly in this section and is described in more detail in chapter 3.

1.3.1 Two CelLEST Reengineering Scenarios

To fully understand and appreciate the motivation behind the CelLEST UI reengineering project, this subsection describes two typical legacy system CUI reengineering scenarios that are most suitable for the application of CelLEST method. These scenarios faithfully represent real cases encountered by practitioners

1.3.1.1 Migrating a Students Information System (SIS) to the WWW

In this scenario, an educational institute developed its student information system (SIS) in the mid-1980s. SIS was written in COBOL and NATURAL (a 4GL) running on the institute's IBM Mainframe platform. SIS included modules like course catalog, schedule of classes, admissions, student biographic data, registration, and academic history. SIS interacts with the Account Receivable Information System, which handles student payments. It also interfaces with a phone registration system. The primary users of SIS are the employees of the Registrar's Office and the students who can only access the registration module using the phone registration system. Additionally, SIS provides system-wide managerial and student information on student enrollment and activity to the administrators to assist them in planning and decision-making on an institution-wide basis. The status quo of the system was quite satisfactory for the management. Its performance, reliability and scalability to high workload at peak times of the year were very good. Additionally, the employees were well trained on the existing character-based UI. System maintainers were familiar with its design and code.

In the late 1990s, the institute wanted to use the Internet to allow students access to student services directly rather than having to go through administrators. Therefore, students could have self-service access to information on enrollment, timetables, grades, and various financial accounts via the WWW. This would free the institute's administrative staff from repetitive and routine tasks and significantly reduce administrative costs.

A UI reengineering solution was needed to open the system to Internet access and to provide an easier alternative to the current character-based UI, which is not easy to use for the general student population. Only tasks relevant to students would be opened to the students. Management ruled out any invasive solution that would involve modifying the existing system due to the cost and risks involved. They also ruled out any solution that would involve duplicating the existing system application logic or data, to avoid possible inconsistencies and extra maintenance overhead. For such a scenario, the CelLEST method is an excellent solution. It can leverage the existing CUI to a Web UI without modifying the existing system. Semi-automatically, CelLEST can generate Web-based wrappers of the desired tasks. CelLEST is an incremental solution that can be implemented and tested gradually by wrapping a small selected number of student tasks and trying them, then wrapping and Web-enabling more functionality, etc.

1.3.1.2 Integrating the Front-ends of Two Insurance Systems

In this scenario, an insurance company acquired another insurance company. Both companies had similar information systems for claims. The performance of each system was satisfactory when they were under different ownership. After the merger, the new owner did not want its employees to use two independent systems, which incurs additional training costs and productivity reduction due to effort duplication and the time consumed in switching between systems. Merging both systems via reengineering or transferring the data of one system to the other was infeasible due to the technical difficulties, prohibitive cost and risk involved. A suitable solution for integration in this case would be CelLEST method. It can provide lightweight front-end integration under a unified task-centered GUI that abstracts both systems' UIs. Additionally, it was required to offer easy access of both systems via an extranet to lawyers who handle cases involving insurance claims. A limited tailored version of the new GUI can be offered to lawyers, which offers specific lawyer-oriented tasks. Since these lawyer-oriented tasks were not required at the time of system development, they were not directly achievable through the existing UIs but the bits and pieces needed for each task were scattered in the character-based UIs. The tailored GUI accomplishes these tasks by taking the necessary inputs from the lawyer in a format that is most natural for him/her and driving the necessary navigation in the two legacy UIs to reach the right screens to gather the needed outputs. Then, it reformats these outputs in the format most natural for lawyers and presents them via the target GUI platform.

1.3.2 The CelLEST Process

Building on the two example scenarios given above, let us now discuss the methodological assumptions of the CelLEST method:

1. The current performance of the legacy system subject to reengineering is satisfactory and it will continue to be used on its current platform.
2. It is required to open the system for access through a new platform, e.g., the WWW or WAP devices or window-based systems, or to integrate the front-end of the legacy system with other applications on the same or other platforms.
3. It is required to take advantage of the presentation and navigation capabilities of the target platform. So, each user task will be encapsulated, with all the input, output and navigation steps required to accomplish it, in a suitable task-oriented representation on the target platform, e.g., a number of Web-forms. In other words, the new UI should be "task-centered".
4. No major functionality change is required. However, minor functionality may be added if it is achievable based on the data presented on the original interface.
5. It is undesirable or impossible to change the legacy system code and architecture. The main input to the UI reengineering process will be recorded traces of interaction between the legacy system UI and its users, while they are doing their regular tasks.

The CelLEST process is a two-phase process. The first is a reverse engineering phase, which is the focus of this thesis [SEKSM99, EISSM01, SEIS03, SES02, ESS02b ESS02c]. The algorithms and methods developed to support the reverse engineering phase and their evaluation are presented in details in this thesis. They are implemented in a prototype tool called the *Legacy Navigation Domain Identifier* (LeNDI). The second is a forward engineering phase, which was conducted by other members of CelLEST research team [KSM99, Kap01, SK02]. The algorithms and methods developed to support the forward engineering phase are implemented in a prototype tool called *Mathaino*.

1.3.2.1 CelLEST Character-base User Interface Reverse Engineering

First, LeNDI is used to record the dialogs that take place between the legacy system and its users while they are doing their tasks in the form of interaction traces. An interaction trace is a sequence of legacy screen snapshots interleaved with the user actions done to cause the transitions between these snapshots. A user action is a sequence of keystrokes. Throughout this thesis, the term "screen" is used to refer to a CUI behavioral state manifested by a matrix of characters displayed to the user on her/his terminal, which allows her/him to do one of a limited set of actions. The term "snapshot" is used to refer to an instance of a screen. One can think of screens and snapshots as classes and objects in object-oriented terms.

Second, for every snapshot recorded in the interaction traces, LeNDI extracts a set of features and employs interactive clustering, classifier induction and user action modeling methods to build a behavioral model of the legacy CUI, called the state-transition graph. The nodes of this model correspond to the CUI behavioral states, i.e., screens and the edges correspond to the user actions causing transitions among the nodes. LeNDI utilizes two clustering methods to group similar snapshots together as one legacy CUI screen modeled by one node on the graph. Then, LeNDI infers a predicate that identifies the snapshots of the screen. LeNDI uses the user actions recorded in the interaction traces to model the behavior of the legacy screens as the arcs of the directed graph. The state-transition graph is a main input to the forward engineering phase of CelLEST. It is used to classify each individual snapshot forwarded by the legacy system to the user while s/he is interacting with it online.

Third, LeNDI uses data-mining methods to discover patterns of frequent segments of interaction between the legacy system and its users which correspond to popular usage scenarios of the system, or in other words the most used services of the system. We call such patterns "interaction patterns". The instances of an interaction pattern may have some noise due to spurious navigation of the legacy CUI. LeNDI interaction pattern mining algorithms can tolerate a preset level of noise and still discover patterns with this level of noise. An interactive review and revision process is associated with the behavior modeling and pattern mining processes. This is to give the user control over these processes on one hand, and to get his feedback to verify the correctness of the models and the usefulness of the patterns produced, on the other hand.

1.3.2.2 CelLEST Character-base User Interface Forward Engineering and Visualization

In the forward engineering phase of CelLEST, Mathaino is used to augment each interaction pattern discovered with the semantic information needed to build a model of the task. Then, Mathaino is used to construct a declarative user-interface specification for the modeled task. This specification is also executable by a suite of special-purpose platform-specific components. Thus, the new user-interface becomes a front-end for the original legacy user-interface, available in multiple new platforms, e.g., XHTML-enabled browsers or WAP devices. The new interface executes a task in the underlying legacy application using the state-transition model of the application's CUI, a model of the task, and an API to the data-transfer protocol used by the legacy system.

In addition to LeNDI and Mathaino, the CelLEST environment includes the QandA (Questions AND Answers) tool [Vij02], which supports the visualization, verification and possibly revision of all the intermediate products of the CelLEST method by the analyst.

1.3.3 Advantages of the CelLEST Process

The CelLEST method is a significant contribution to the field of CUI reengineering. This is because it has a number of advantages:

1. It is a code-independent non-invasive CUI reengineering method. It utilizes easy to collect input, i.e., interaction traces. So, it is very suitable when code modification is undesirable, expensive, risky or impossible. The limitation of this method is that it can support only minor functionality extensions.
2. CelLEST is lightweight in terms of the skills it assumes. It needs moderate analysis skills and an understanding of the system under analysis as opposed to the high software development skills and expert understanding of the legacy system that current practices demand. It is lightweight in terms of the cost and time. Therefore, it can potentially bring substantial time and cost reduction and quality improvement to current state-of-the-art industry practices.
3. CelLEST is an incremental approach. It can be applied gradually to the exiting legacy CUI. Thus, a phased reengineering can take place with some services of legacy system CUI reengineered in every phase, according to the time and budget available.
4. CelLEST follows a task-centered approach to reengineering the way the legacy system users interact with it. It encapsulates interesting behavioral segments within new UI front-ends on different platforms. It does not replicate the legacy system-user interaction with different widgets in new platforms.
5. CelLEST supports simultaneous migration to multiple platforms. It does CUI reverse and forward engineering once and generates abstract specifications of the

reengineered task-centered UI. A new UI can be generated multiple times on different platforms using the abstract UI specifications.

6. CelLEST emerged from collaboration with industry on one hand and on the other hand, it was developed and evaluated in an academic setup, with very encouraging results. This mixture of research and industry gives CelLEST the potential to impact current CUI reengineering practices on solid scientific bases.

1.4 Thesis Statement

This thesis makes a case for automated CUI interaction reverse engineering. It takes the position that recorded traces of interaction between the users of a legacy system with its character-based user interface can be sufficient input for lightweight non-invasive reengineering of the user interface. It demonstrates that reverse engineering these interaction traces can provide the CUI behavioral model required for the reengineering process and shows how the elements of this model can be inferred from these traces. Additionally, it demonstrates that patterns of user activities with the legacy CUI can be discovered from the interaction traces, and used as a basis for identifying and modeling the system services that are candidate for reengineering. Finally, this thesis demonstrates, via case studies, the practicality, efficiency and usefulness of the automated CUI interaction reverse engineering process, and hence, proves its potential impact on advancing the current manual practices for lightweight CUI reengineering and Web-enabling.

1.5 Thesis Contributions

This thesis establishes a novel method for CUI reverse engineering that adopts interaction reverse engineering as the means to build a behavioral model for the legacy CUI and to capture the interesting user interaction patterns with the system. The behavioral model and the interaction patterns are used for CUI reengineering. The specific contributions of this thesis are:

1. **Engineering a suite of features for characterizing CUI screen snapshots [SEKSM99, SEIS03].** These features are extracted from analysis of the snapshots with a set of tailored heuristics and document analysis methods. This analysis extracts features from any special information discovered at the periphery of the snapshot,

from the hidden snapshot information coming with the outbound data streams received from the host, and from the snapshot layout and content distribution. Some of these features are specific to IBM 3270 data streams, but most of them are applicable to other block-mode data transfer protocols, which push one screen at a time to the user (as opposed to scroll-mode data transfer protocols, which interact with the user line by line).

2. **An intelligent semi-automated method for modeling the behavior of legacy CUIs [EISSM01, SEIS03]**. This method is a significant advance to the research and practice of CUI reengineering. It consists of the following steps:
   - Recording traces of interaction between the legacy system CUI and its users.
   - Extracting a feature vector for every recorded snapshot.
   - Clustering similar snapshots together, based on their feature vectors similarity.
   - Inferring predicates for each cluster (i.e., each distinct CUI state) via classifier induction.
   - Example-based learning of the user actions that cause transitions from one state to another.
   - Mining the interaction traces for patterns of user interaction with the legacy CUI.
3. **Two novel sequential pattern mining algorithms [ESS02b, ESS02c].** IPM is a breadth-first algorithm and IPM2 is a depth-first algorithm. Although, both algorithms are designed specifically to mine interaction traces for interaction patterns, they can mine sequential data in general for sequential patterns with noise.
4. **A prototype tool for interaction-based CUI reverse engineering, called LeNDI [SES02, SEIS03]**. LeNDI implements all the methods and algorithms described in this thesis. LeNDI was used to evaluate the CUI reverse-engineering method with case studies from real legacy systems, with very promising results.

1.6 Thesis Organization

The rest of this thesis is organized as follows:

- Chapter 2, *Related Work*, introduces the related research areas. It covers four areas of research that represent the broader areas of this thesis and the state-of-the-art in research and industry of UI reengineering and Web-enabling.

- Chapter 3, *CelLEST User Interface Reengineering*, is a detailed description of the CelLEST method for CUI reengineering at large. It helps the reader understand the context, in which, the research described in this dissertation was conducted.
- Chapter 4, *Feature Extraction For Legacy Screen Snapshots*, describes the feature suite engineered in LeNDI for characterizing the snapshots of CUI screens. It details the process of extracting a feature vector for every recorded snapshot and the algorithms used in it.
- Chapter 5, *Legacy User Interface Behavior Modeling*, describes the process and algorithms used to identify the nodes and arcs of the state-transition model of a legacy CUI, and consequently build it. Additionally, it discusses the experiments and case studies conducted to evaluate this process and the associated algorithms.
- Chapter 6, *Mining Interaction Traces for Patterns of Frequent User Tasks*, describes the process and the two novel algorithms (IPM and IPM2) used to mine the interaction traces for interaction patterns. It provides evaluation for this process and comparison of the performance of IPM and IPM2.
- Chapter 7, *Summary, Conclusions and Future Work*, presents a summary of this dissertation, draws some conclusion and points to future work directions.

Chapter Two
Related Work

This chapter describes four areas of research related to the research described in this thesis. The diversity of these areas reflects the diversity of the methods and algorithms used in this research. The first area is *UI reengineering and reverse engineering*. The second area is driven by practice and industry more than by academia, which is *Web-enabling legacy systems*. The third area is *software requirements recovery* from available legacy resources. The forth area is *data mining of sequential data* to discover sequential patterns in long sequences of data. Each of the four sections of this chapter is dedicated to one of these areas. Each section starts by a definition of the research area, followed by the motivation behind studying this area and by a description of representative, state-of-the-art work. Finally, it concludes with comments on how this work is similar to or different from the work in this thesis.

Section 2.1 is a review of the work in the area of UI reengineering and reverse engineering. The CelLEST project is about UI reengineering, but is different than other work in this area in that it adopts a lightweight "don't touch code" approach for UI reengineering. LeNDI adopts a novel UI reverse reengineering method using interaction traces, designed to suit the interaction reengineering approach of CelLEST

Section 2.2 describes the state-of-the-art industrial practices of Web-enabling legacy systems, by uncovering and accessing the logic, data or presentation layers of the legacy system. In particular it focuses on the current manual practices of reverse engineering legacy CUIs using presentation layer access [Ake00, Att00, Cri01] and describes how LeNDI supersedes and advances these manual practices by providing a coherent automated process that is more efficient and less susceptible to error. The area of Web-enabling legacy systems is described in more detail since it is the most related area to this work. In addition, since a lot of Web-enabling work is coming from industry not academia, detailed discussion of the evolution of this area, that relates different practices to each other, was needed.

Section 2.3 describes the related work in the area of software-requirements recovery. It describes the different methods used or proposed in literature for recovering the functional requirements or specifications of software systems, using different inputs. LeNDI employs a novel automated method for recapturing models of the current uses of a legacy system as its *de facto* functional requirements. LeNDI's method is easy, practical and does not assume the availability of system code or documentation. While the recovered models are needed for the CelLEST forward engineering phase, they can be translated to use case models and utilized in different contexts.

Section 2.4 describes the related problems and significant work in the area of sequential pattern mining, inspired by data mining and bioinformatics applications. It explains the problem of interaction pattern mining in LeNDI, which is mining the traces of interaction with a legacy system for interaction patterns. It describes how this problem is different from other problems and hence, why LeNDI needs a new algorithm to solve it. This led to developing two novel algorithms for interaction pattern mining, Interaction Pattern Miner (IPM) and Interaction Pattern Miner 2 (IPM2), which are described in details in chapter 6.

2.1 User Interface Reengineering and Reverse Engineering

UI Reengineering is the process of recreating an existing UI, either on the same or a different platform. In some cases, UI reengineering is done as part of legacy system migration to a new platform. In other cases, UI reengineering is done for itself to reface an existing legacy application with or without minimal changes to the system.

UI Reengineering is a two-step process. The first is a reverse engineering phase, during which an abstract representation of the legacy UI is created. The second is a forward engineering phase, during which, a new or modified implementation of the legacy UI is developed, usually on a new platform. Reverse engineering an existing UI can be desired in order to better understand an existing system. In such cases, the objective is to produce an abstract representation of the legacy UI to enhance the understandability and maintainability of the legacy system, especially its presentation layer. LeNDI is a CUI reverse engineering tool that uses, as input, recorded traces of interaction with the legacy system. Mathaino is a CUI forward engineering tool that uses the models developed by LeNDI to develop new GUIs, Web-interfaces or WAP-interfaces for legacy systems. In developing LeNDI, the significant work on UI reengineering and reverse engineering was reviewed as follows shortly. None of the work described adopts the novel approach of interaction reverse engineering employed in LeNDI. LeNDI avoids code analysis and takes advantage of the relatively easy to collect interaction traces to deduce the legacy UI behavior models and interaction patterns needed in CelLEST.

Reengineering UIs of legacy systems can have different variants:

- UI full or partial redesign and re-implementation on the same platform [PRSV97].
- GUI to GUI migration due to platform change [MRS94].
- Character-based UI (CUI) to GUI migration [AFMT95].
- GUI grafting onto a batch or command-line system [PA97, TS99, WJD01, SCT02].
- Wrapping a legacy UI with a Web or WAP wrapper [Hor98, TLRH98, BFM02].

An example of the first category is the work of Plaisant *et al.* [PRSV97]. They employed different techniques to evaluate existing UIs for six different systems. These techniques include documentation study, observing users, expert review of the current UI, questionnaires and discussions with users and managers. For each system, they identified the main problems in its UI, if any, and the opportunities of improvement in user documentation, system access, data display, data entry procedures, consistency and error/system messages. These findings were used to partially re-implement the examined UIs on the same platform to improve user performance and satisfaction.

An example of the second category is the work of Moore *et al.* [MRS94]. Their approach relies on a knowledge-based model that maps the functionality of the widgets in the source platform UI toolkit to those of the target platform UI toolkit. Given a particular migration problem, the software engineer identifies the pieces of code in the system implementing the UI. Then, based on the knowledge-based model, a "wizard" guides the software engineer in selecting appropriate widgets in the target platform toolkit that can together deliver the interactive behavior of the original code.

A similar approach was used to address the third category [AFMT95]. The interesting difference is that since there is no source widget toolkit, the reverse engineering process hypothesizes widgets from the code.

Grafting a GUI on top of a relatively non-interactive set of batch or command-line programs (CLPs), whose functionality is accessible only through command-line inputs, is simpler and does not involve a complex reverse engineering problem. The GUI can be developed independently, with the aid of a GUI-building tool that uses an abstract representation of the underlying applications. Then, the user-initiated events on the GUI are programmed to invoke procedures in the underlying application programs [PA97]. To enable a degree of freedom in utilizing the underlying programs and formatting their outputs, some source code reverse engineering might be required in the beginning to identify internal variables and data structures of interest in order to expose some of them in the new UI [TS99]. Moreover, to allow flexibility in output formatting or to integrate with distributed object middleware, e.g., CORBA, a wrapper can be placed around the command-line application to programmatically invoke its commands. Then, the wrapper parses the generated output and returns a semantically useful result (an integer, an object, etc.) that can be easily consumed by a calling program or a GUI object [WJD01].

Sorzano *et al.* [SCT02] present a model for CLP packages. The model includes a command-line syntax specification, which is integrated into a higher-order OO model that can be directly translated into a graphical user interface. The object types (classes) in this model are: package, group, program, command line, menu and argument, where a package is a CLP application, a group is a subset of related programs in the package and a menu here is a list of arguments. The authors described a language, Colimate (the COmmand LIne MATE) that implements this model. Using Colimate, a GUI description can be written for legacy CLPs and then compiled and run with the help of an interface generator that raises the needed windows, attends to user selections and finally launches the desired processes.

Finally, work in the fifth category, i.e., wrapping with a web-enabled wrapper, is one of few possible solutions to the broader problem of Web-enabling legacy systems, which is discussed in the next section.

Dannelly [D95] presents a case where UI reverse engineering is an objective for itself. He introduced methods for automatic analysis of the source code of X Window System application programs and transforming it into an intermediary representation. This representation is used for automatic production of two types of graphs. The first is widget-instance trees, which are inferred from initialization code and represent the hierarchy of widgets created by this code. The second is dialog-state diagrams, which are graphical representations of the behavior of X Window System based GUIs and represent the action routines associated with the different widgets created in the initialization code. These graphs improve the system maintainers understanding of the legacy system UI and ease their job.

Except from few examples, the majority of UI reengineering and reverse engineering approaches adopt code analysis and understanding as the means for system modeling and reverse engineering. However, the UI-related code is only part of the system code that has to be examined. Additionally, due to ageing symptoms mentioned earlier, it is hard and expensive to analyze the legacy code. Even worse, in some cases the code is not even available.

In such cases, system-user interaction can be an alternative source of information for understanding the legacy system. It is a rich source of knowledge and a faithful representation of how the system is currently being used. Hence, it is a good candidate input for the UI reengineering process. Interaction reverse engineering in LeNDI is novel and different than the work summarized above in that it uses this input instead of source code for producing the necessary abstractions and models for the consequent forward engineering phase. This reengineering method does not alter or change the existing legacy system code or structure. It relies on the almost-automated process of LeNDI to infer most of the CUI behavior and task models it needs. As described in chapters 5 and six. These models are useful beyond CelLEST and can be deployed in other contexts as described in section 7.3.

2.2 Web-enabling Legacy Systems

Web-enabling legacy systems is the process of opening an existing legacy software for access through the Internet, an extranet and/or an intranet to the public and/or to a selected user-base of employees, customers and/or business partners. The work in this area is mostly industry driven. It is motivated by the emergence of the Internet as a medium for doing business, with new opportunities of business growth and cost reduction. Using the Internet, especially the WWW, a company can reach out to more customers worldwide via a simple, easy to use, inexpensive and very popular UI, namely the web browser [BFM02]. Additionally, via their web sites, businesses can reduce the cost of transactions and customer service by automating their transactions and moving the effort of customer service to the customers themselves via online orders, quotations, etc.

Since somewhere between 60% and 80% of all corporate information reside on mainframe-based legacy systems [Att00], opening many of these systems for Web-access may be essential for corporate Web-enabling strategies. A number of solutions to this problem were developed mostly by industry, which may require reproducing legacy business data and business processes in new formats and new presentations for old and new audiences [Lan00]. These solutions complete one another in some situations, and compete with one another in other situations. But, in practice a variety of Web-enabling technologies may be used, even within the same corporation. This is due to the wide variety of legacy and Web technologies available and the unique requirements of every Web-enabling project. This section briefly describes these solutions and their advantages, disadvantages and limitations. A legacy software application in general, consists of three layers: the presentation layer, the program-logic layer and the data layer. A Web application can access a legacy system via one or more of these layers, depending on the available legacy and Web technologies and the status of the legacy system.

Increasingly, Web-enabling is becoming one of the main activities in the area of UI reengineering. Web-enabling is also one of the main uses of the CelLEST method for legacy CUI reengineering. Interaction reverse engineering in LeNDI is a significant advance to the practice of modeling existing legacy CUIs for Web-enabling via presentation access. It provides a coherent almost-automated process for modeling legacy CUIs in preparation for Web-enabling that replaces the current manual error-prone time-consuming practices.

2.2.1 Web-enabling via Data Access

In data access Web-enabling, a new Web application is developed to directly access the legacy database and then perform the necessary processing on the retrieved data before presenting it to the user via a web browser. The primary assumption is that, the legacy application logic is trivial and can be easily duplicated in the Web application if required or that new logic will be developed to process data different from the legacy logic. The legacy data must be wrapped in order to be accessed using a different interface or protocol than that for which the data was designed initially. This requires using data access-middleware in the form of database *gateways* and *bridges*. A database gateway is a software application that translates between two or more data-access protocols. There are a number of *de facto* industry standard database gateways. Open Database Connectivity (ODBC) is Microsoft's interface for accessing data in a heterogeneous environment of relational and non-relational database management systems. The ODBC API can be invoked inside Active Server Pages (ASP) or programs in C/C++, Perl, VB, etc. Java Database Connectivity (JDBC) is an industry standard defined by Sun for database-independent connectivity between Java applets or applications and a broad range of SQL databases. The JDBC API can be invoked inside a Java program, applet or servlet or a Java Server Page (JSP). ODMG is the standard of the Object Data Management Group for persistent object storage. A bridge is a special gateway that translates one standard protocol into another, e.g., JDBC-ODBC bridge. [CWSR00]

Web-enabling via data access is a simple, straightforward solution [Amb00]. It can provide multi-source legacy data integration to new applications [RMB00]. It has some considerable drawbacks. First, any important logic, e.g., business rules and data validation, is bypassed and not utilized. This means that it has to be re-implemented in the web application. This may cause duplication in program logic and high cost in both development and maintenance. Second, it increases the data coupling between the legacy and web applications [Amb00, RMB00]. This solution is most suitable for Web-enabling legacy services with trivial logic, e.g., Web-enabling the catalog browsing services of a legacy library system. It is also suitable, when new logic needs to be implemented to process the data for Web-access, so, this logic can be implemented in the web application.

Data replication is another data access based Web-enabling method. In this case, part of the central legacy database is duplicated on a web server for Web-access through a client application. The legacy and server data repositories are coordinated with periodic batch jobs. The major weakness of this approach is that it cannot handle real-time data, making it inapplicable for applications such as customer service or sales activity.

2.2.2 Web-enabling via Logic Access

Web-enabling via application logic access relies on the availability of a mechanism to access the business logic independent from the user interface related code. This can be accomplished in different ways:

Code Access. If the legacy business logic is implemented separately from the presentation logic, which is rarely the case, then theoretically, it is possible to insert a thin control layer that accepts the data extracted from the HTML page and invokes the appropriate subroutine from the legacy system. The business-logic subroutine processes the request and gives back the results, which are then forwarded to the client [Sne00].

API Access. Packages like SAP, PeopleSoft, etc., offer APIs that can be accessed via Java Native Interface (JNI) or Common Gateway Interface (CGI) code. But, software developed in-house rarely has a defined API, or at most it has a very limited function-oriented (not OO) API. [Amb00]

Distributed Object Technology (DOT) extends object technology to the net-centric information systems of modern enterprises by using object middleware, e.g., OMG's CORBA and Microsoft COM+. The idea is to *objectify* (or *objectize*) the legacy system by creating an OO interface to individual applications, common services and business data that makes the legacy software look like objects. Then it can be accessed by other applications across a network through the OO interface. [CWSR00, RMB00, ZK99]

This is quite an invasive reengineering solution. The prime challenge is objectifying the legacy system, i.e., analyzing, decomposition, and then translating the monolithic and plain semantics of the usually procedural legacy system to the richly hierarchic and structured semantics of an object-oriented system [CWSR00]. Several methods were suggested to decompose legacy systems into objects, including cluster analysis, concept analysis and hybrid methods [CCDD01, PZKM99]. The amount of effort needed to accomplish this task depends on the language, style and architecture used in developing the legacy application.

Component wrapping is a natural extension to DOT. In contrast with objects, components must conform to a component model. This constraint enables the component framework to provide the component with quality services [CWSR00]. Enterprise JavaBeans (EJB), from Sun Microsystems, is an example of server-side component architecture for writing reusable business logic and portable enterprise applications. EJB is the basis of Sun's Java 2 Platform, Enterprise Edition (J2EE). EJB components are written entirely in Java and run on any EJB compliant server. Each bean encapsulates a piece of business logic. EJB servers provide system-level services such as transactions, security, threading, state management, resource pooling, distributed naming, remote invocation and persistence. [CWSR00, FOLD96]

EJBs can wrap exiting legacy system functions and offer them as operating system and platform independent components. But like DOT, component wrapping faces the challenge of *componentifying* the legacy application, which is separating the interface of the legacy system into modules consisting of logical units or functions. [CWSR00]

Zou and Kontogiannis [ZK99] describe a combination of DOT and component wrapping technology. First, they identify and generate a decomposition of the legacy system into modules, and then analyze the interfaces of the selected legacy components and store their signatures in a component repository using XML format. Second, they generate the CORBA/IDL and CORBA wrappers from the component repository. Third, they use EJBs to develop the application server, in order to integrate the CORBA wrappers and to provide the services to the Web-based application. Finally, they define a scripting language using XML, to enable the invocation of the legacy components.

2.2.3 Web-enabling via Presentation Access

Web-enabling via presentation access is non-invasive and almost risk-free. It covers a wide spectrum of solutions. At one end, there is Web-enabled emulation of legacy systems. At the other end, there is screen mapping, which allows complex manipulations of the legacy data streams used for communication between the legacy host and the legacy terminals. Thus, it supports crafting reengineered front-ends for legacy CUIs that take advantage of the potential of Web UIs. The common attribute of all these approaches is that the legacy application is accessed via its presentation layer. This can be done mostly by accessing its UI, represented by the legacy outbound and inbound data streams, or in other cases by accessing its presentation description if one exists, e.g., IBM Customer Information Control System (CICS) maps. In both cases, the access is limited to the data and operations offered via the application presentation.

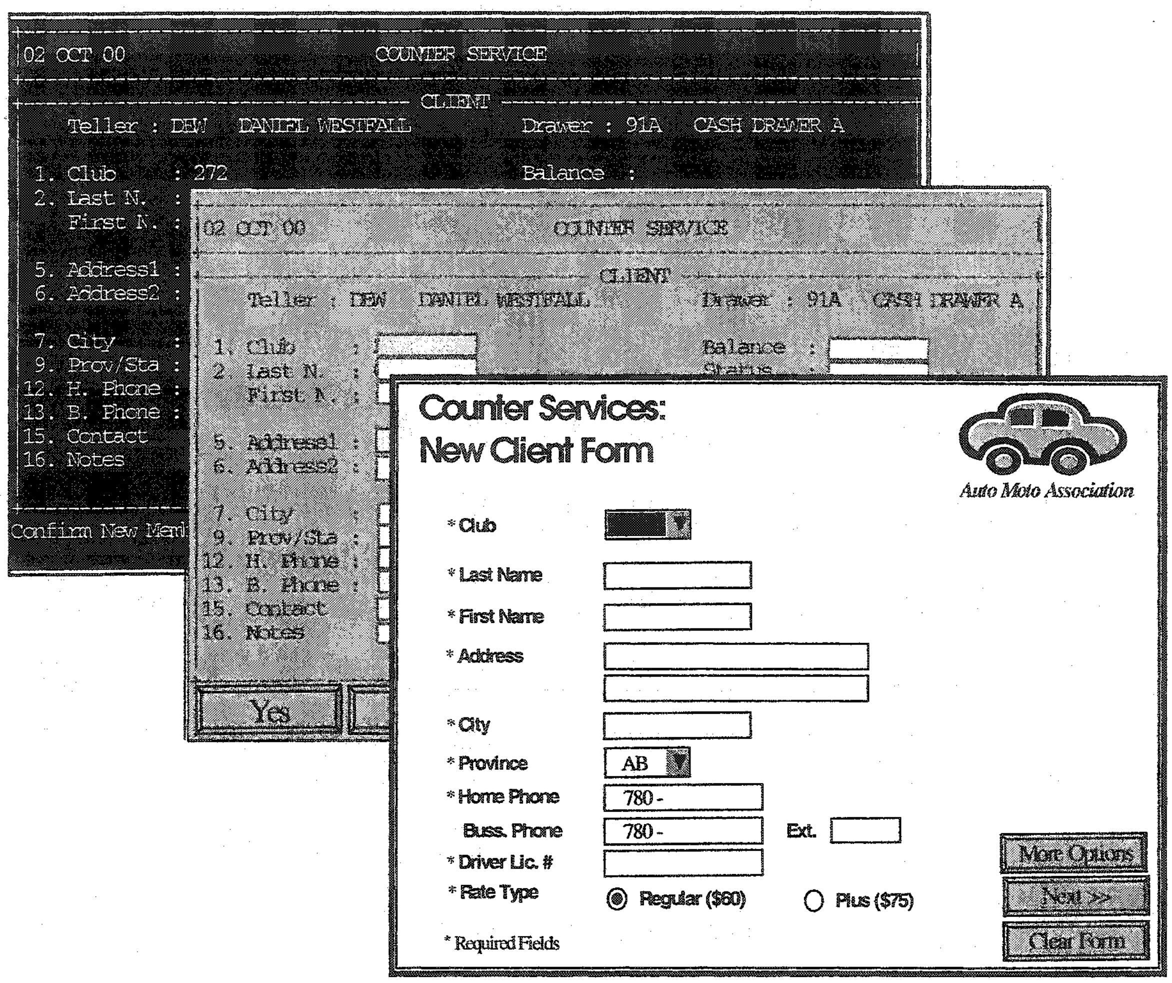
Web-enabling via presentation access started in its simple form of web emulation (webulation [BB01]), shortly after the emergence of the Internet [TLRH98]. Gradually, webulation evolved into the more advanced screen scraping technology that takes advantage of the GUI capabilities of web browsers. The next generation was manual screen mapping that allows remodeling the legacy UI or parts of it into a task-oriented

Web-based GUI. The future trend, which was the subject of our research in CelLEST project, is to automate screen mapping as much as possible, using intelligent tools that can learn about the legacy system on its own. Then, the knowledge learned is used to build the models needed for screen mapping with minimal effort. Consequently, these models are used to build a working Web-based GUI or abstract UI specifications that would allow simultaneous migration to multiple platforms, e.g., the WWW and WAP devices. Additionally, screen mapping is extended to allow the integration of multiple legacy systems UIs together or with other Web applications under a unified Web front-end. In the following we discuss the evolution and available solutions for Web-enabling via presentation access.

2.2.3.1 Web Emulation (Webulation)

Web emulation, or webulation [BB01, BFM02, TLRH98], is a natural extension of the long-practiced legacy host emulation to the Web. The new thing is that the emulator runs in a web browser or a web server. Browser displays have the native look and feel of the host legacy screens. Transactions work exactly the same as on a legacy host "green screen" terminal, e.g., IBM 3270, by returning one screen display for one input request. Full support for legacy function keys as well as user customization of colors and fonts are usually available. Additionally, icons for function keys, copy/paste, macro recording, file transfer and other basic operations are provided. [Ake00, BFM02]

Webulation is a quick and cheap solution that does not need any Web application development. It offers instant access to the legacy application to intranet and extranet users who are already familiar with the legacy system. But, for the wide Web population or new users, it does not make the legacy system any easier to use [BFM02]. Additionally, it does not allow tailored Web-access that targets different groups of users with different limited sets of UI functionality. It addresses the accessibility issue, but it does not improve the usability or navigability of the legacy system (see section 1.2). A typical implementation of webulation is done using Java applets, downloaded into the client web browser. The applet runs in the web browser Java runtime environment and establishes a connection with a Telnet server that manages access to the host application. [BFM02, TLRH00]



```
02 OCT 00                         COUNTER SERVICE
------------------------------- CLIENT -------------------------------
   Teller : DEW   DANIEL WESTFALL          Drawer : 91A   CASH DRAWER A

 1. Club      : 272                        Balance  :
 2. Last N.   :
    First N.  :

 5. Address1  :
 6. Address2  :

 7. City      :
 9. Prov/Sta  :
12. H. Phone  :
13. B. Phone  :
15. Contact
16. Notes

Confirm New Mem
```

```
02 OCT 00                         COUNTER SERVICE
------------------------------- CLIENT -------------------------------
   Teller : DEW   DANIEL WESTFALL          Drawer : 91A   CASH DRAWER A

 1. Club      :                            Balance  :
 2. Last N.   :                            Status   :
    First N.  :

 5. Address1  :
 6. Address2  :

 7. City      :
 9. Prov/Sta  :
12. H. Phone  :
13. B. Phone  :
15. Contact
16. Notes

[Yes]
```

Counter Services: New Client Form

* Club		
* Last Name		
* First Name		
* Address		
* City		
* Province	AB	
* Home Phone	780 -	
Buss. Phone	780 -	Ext.
* Driver Lic. #		
* Rate Type	[✓ Regular ($60)]	Plus ($75)

* Required Fields

Figure 2.1. An Example Legacy Screen (upper), Refaced On-the-fly (middle) and Refaced Using Screen Customization (lower)

2.2.3.2 Screen Scraping (Refacing)

Screen scraping (or refacing [Ake00, Att00]) takes webulation a step further by offering an enhanced *one-for-one* browser presentation of the legacy UI. It reads the data stream intended for the mainframe terminal, either via a client based terminal emulator (Java applet) or a server based program, and turns it into a Web-based GUI presentation. The translation of each legacy screen to a Web-based GUI can be done in two ways, either on the fly or using a user defined customization for this screen. Figure 2.1 shows both cases.

In the first case, a middleware is interleaved between the Web front-end and the legacy software to act as a presentation translator by intercepting outbound legacy displays and converting them "on-the-fly" into a Web front-end using whatever available information. In case of IBM 3270 for example, this is done by converting the unprotected (input) fields into edit control objects while turning the other fields into labels. Some advanced tools convert a set of predefined strings like F1,... F24 into buttons. Slight improvement is achieved over webulation.

In the second case, an individual customization is created for every screen that radically changes its appearance and takes advantage of the presentation potential of the target platform, the Web in this case. Thus legacy screens appear "dressed up" in a Web-based GUI, with widgets, lists, radio and push buttons, images, web links, check and choice boxes, colors, fonts, etc. Additionally, one can reorder fields, change tab sequence, and hide unnecessary data [Ake00, BB01, BFM02]. Figure 2.1 shows a legacy screen that is refaced on-the-fly (middle) with unprotected (input) fields turned into text boxes and "confirm new member ? (Y/N)" message replaced by "Yes" and "No" buttons. The same screen is refaced with individual customization that turned the original screen fields into text boxes, lists, radio buttons, etc., wherever suitable. Also, a logo and a big font title were added. A few buttons with additional functions were added.

Some screen scraping tools, e.g., IBM Screen Customizer [IBM99, BFM02], offer the ability to create context-sensitive field help for host applications, to create a list of valid values for a data field and/or to skip unnecessary screens during navigation. Also, they allow customization templates to be applied, in order to speedup refacing a number of legacy screens. Such a template would contain common elements to all Web-based screens replacing the legacy screens, e.g., a logo, a web link, a customized tool bar, etc. The operation of the applications is still "one-for-one". That is, one browser request equals one legacy screen display [Ake00].

To know which customization to apply to a screen snapshot, the screen scraping application should recognize the identity of the instance. This is done using a predicate or signature for every screen. Typically, this signature is based on some unique keyword(s) that appear on the screen at some location(s). Some tools offer rich pattern definition languages for the application builder to define a signature [BFM02, Cel99]. Such languages would allow specifying that a specific text must exist or not exist at a certain location or within an area on the screen and/or that it must be of a certain case or color, or compares in a certain way to a hard-coded value (<, <=, =, >=, >). They may also allow multiple recognition criteria to be defined and combined with logical operators, for the same screen.

In many cases, this technique would be sufficient to recognize the identity of the screen. However, in some cases, e.g., unstructured and multi-mode screens, it is quite challenging to find such a signature. For example, some host applications can have more than one mode for the same screen, e.g., Create, Review, or Update modes, with the same structure and appearance but with slight differences in the status of some fields. Each mode needs separate signature and customization. In the state-of-the-art practices, a screen signature must be manually defined and hard coded for every screen by an expert analyst, who is very familiar with the legacy system under analysis and with the pattern language available and its supporting tool.

Screen scraping takes relative advantage of the presentation capabilities of the web browser. However, it does not benefit from its enhanced navigability. So it enhances the accessibility and, to some extent, the usability but not the navigability of the legacy system.

Another screen scraping approach is to access the legacy presentation at a level lower than the UI, or the data streams used to construct it. This is the level of screens maps or description files, if such concept exists in the system under study, e.g., CICS maps for S/390 systems and Data Description Specifications (DDS) source files for AS/400 systems. In case of CICS, the data necessary to build an HTML or XML UI is extracted from CICS maps instead of IBM 3270 data streams.

2.2.3.3 Screen Mapping (Remodeling)

Screen mapping [Cri01], also called remodeling [Ake00], is a natural extension to screen scraping. While it still uses presentation access to Web-enable legacy systems, it allows reengineering the legacy UI or chosen parts of it into a task-oriented Web-based GUI. It enables fairly extensive modifications to the sequence of information presented to the user by combining several screens into a single graphical presentation, i.e., it offers *many-for-one* browser presentation of the legacy user interface [Att00]. Thus, the multiple host screens, related to a certain user task are combined in one (or more) Web forms that is a more natural representation of the task in the Web world. This can greatly enhance the usability and navigability of the system, while still maintaining the back-end host navigational flow. It is possible to partially apply screen mapping to a legacy system by reengineering the frequent or cumbersome user tasks of choice, while using webulation and/or screen scraping for the rest of the system.

To build a screen mapping solution, one needs to do the following:

1. Build a model of the portion of the legacy system CUI to be reengineered,
2. Describe the steps needed in terms of user actions, inputs and outputs, and screens accessed to perform each user task that will be reengineered,
3. Build/buy the middleware needed to mediate between the legacy back-end and the Web front-end, and
4. Build a Web-based GUI for each user task, which will be responsible of executing the task plan via the host-access middleware.

The partial model built for the legacy CUI is a set of predicates or signatures that should uniquely identify each legacy screen, along with a list of the possible behaviors of each screen in terms of the user actions permissible on it and their outcomes or destination screens. This model is like a road map for the legacy CUI.

A task description gives the detailed steps of how to open sessions, gather data, complete transactions, and close sessions with the host legacy to accomplish a user task. Typically, this includes what user actions are needed to navigate the legacy CUI in service of the user task, what inputs need to be passed to the legacy application on which screens on which locations and what outputs will be retrieved from which locations on which legacy screens.

A host-access middleware executes the task description by driving the necessary navigation via the legacy host, passing the user inputs received from the Web-based GUI to the legacy application, and collecting the required outputs to feed the Web-based front-end. This middleware uses terminal access protocols such as VT100, IBM 3270 or IBM 5250 to communicate with the legacy system via a “virtual terminal”, emulating the standard “green screen” terminal. Data are moved in and out of the legacy host via the legacy system CUI as if data entry personnel were flawlessly entering it [Cri01]. Such middleware can be built with EJB beans, Java servlets, or similar approaches.

The Web-based GUI, e.g., HTML or XHTML, presents the reengineered UI to the user, takes his inputs, executes the task plan and reproduces the collected results through the web browser. Common Gateway Interface (CGI) scripts, Java Server Pages, or similar technologies are used to collect the user input via the client web browser

In current practices, these steps are all implemented manually. An analyst goes through each screen of the legacy system, trying to find a unique signature for it and to model its behavior in order to build the legacy CUI model. Then s/he talks to the users about every task to be migrated to the Web to understand all its possible paths and exceptions. Then, s/he manually builds a plan of this task, by describing all the user actions needed to perform it, all the inputs to be entered and where they occur and all the outputs to be collected and from where they are obtained. After that, a developer builds the new Web front-end that executes the user tasks. For each task, s/he does the necessary GUI design, layout and coding. If it is required to migrate to a different platform too, e.g., WAP devices, then the legacy CUI model and task descriptions can be reused but the new UI implementation should be carried out from scratch for the new platform.

2.3.4 Pros and Cons of Web-enabling via Presentation Access

A market survey [Att00] showed that 60% of the IT personnel administrating, maintaining or accessing information from legacy systems use some form of screen scraping or screen mapping technology to integrate legacy systems with other systems. 45% of them batch data to a server for access through a client application, and a similar number modify the host application to suit client access. The study showed that out of those who use presentation access technology, 60% use it to avoid changing the host application, 30% could not change the host application and 44% use it for its lower cost.

Despite the possible bias in market studies, these results summarize the advantages of Web-enabling via presentation access. It is a minimal-risk, non-invasive, less expensive solution. It requires no change to the legacy application. This makes it almost the only choice when changing the legacy application is not an option, e.g., due to lack of ownership or unavailability of the source code. It can be applied gradually and/or using a mixture of methods. For example, webulation can give instant Web-access of the legacy application to the users familiar with it, while a screen mapping solution is deployed to reengineer the UI of the most frequent/difficult user tasks. This provides an easy to use HTML front-end to the external users with no familiarity with the legacy system, e.g., customers placing orders and college students registering for classes. Finally, presentation access can be used for lightweight integration with other legacy systems and Web applications or for limited functionality extension and repurposing of the legacy system.

On the other hand, screen scraping and screen mapping are labor intensive non-automated processes. The currently available tools for supporting them are mostly limited to aiding the manual practices. They do not automate any of the subtasks involved, which may require a lot of effort and intuition. Additionally, these technologies are better suited to mature stable applications, which are unlikely to go through frequent changes or updates. For dynamic applications that go through frequent changes, keeping the Web application layer up to date with the latest changes incurs high maintenance overhead.

Presentation access of legacy systems is criticized for being slow, since it adds a remote extra layer on top of the existing legacy application. This depends on the implementation model used. Modern server side and host side implementations can overcome this deficiency to a good extent. In a host side implementation, the host-access and the screen mapping middleware reside on the legacy host, e.g., S/390. A user task is executed completely on the host and HTML pages are generated as needed and submitted to the user with the required results or to collect inputs. However, fair comparison with the other Web-enabling approaches presented earlier is unavailable to judge their relative speed and scaling up with workload.

Another disadvantage of presentation access solutions is their vulnerability to unexpected events related to the host connection behavior, like keyboard lockups, session disconnections, broadcast messages from hosts and error messages coming from the legacy application [Yam00]. Careful analysis and modeling of the legacy CUI and the tasks to be reengineered can reduce this risk by anticipating as many of such events as possible and including a recovery mechanism in the Web front-end application, but would require more investment and effort.

Finally, Web-enabling via presentation access has some limitation: it cannot extend the legacy system functionality beyond what is already achievable, directly or indirectly, through the legacy presentation. It only gives access to the data exposed through the legacy presentation.

The CelLEST project adopts presentation access for legacy CUI reengineering and Web-enabling. LeNDI overcomes some of the disadvantages of presentation access solutions by providing a coherent almost-automated process and tool support for interaction reverse engineering and legacy CUI modeling. LeNDI uses traces of interaction with the legacy CUI as input. LeNDI provides a data mining method that discovers the frequent user tasks of interest in the form of interaction patterns, as they are evident in the interaction traces. These patterns are used to build the task models that will be encapsulated in the reengineered UI. LeNDI reduces drastically the level of skills, time and cost needed for the modeling process, since its subtasks are almost automated. Additionally, the automated process is less sensitive to changes in the underlying CUI as changes can be captured by recording and analyzing new or extra traces instead of implementing them manually. LeNDI advances the current screen mapping practices and lays the foundation for the future generation of these solutions.

2.3.5 Objectifying Legacy Systems via Presentation Access

Little work has been done along this somewhat different line, which mixes presentation access of legacy systems with distributed object technology. There is very little research in this area, and it suggests screen scraping of legacy CUIs, or similar techniques for legacy GUIs, as a means for wrapping a legacy system service (or a user task) as a method in an object for consumption by a new application or in a distributed object environment.

Chadha [Cha98] describes a prototypical effort, during which, the services of several back-end legacy systems belonging to various health insurance providers were integrated in one distributed object environment to provide access to these systems to healthcare providers. This was done via a distributed object, called the "Payer" object. Each Payer object wraps a legacy data source, e.g., an IBM mainframe application, an ODBC-enabled database or others. The Payer object interface allows healthcare providers to check the eligibility of patients for insurance coverage, submit insurance claims, and check the status of submitted claims. The object interface is described using CORBA IDL. For IBM mainframe-based legacy back-ends, the Payer object uses screen scraping to invoke the procedures that perform the required services on the legacy application and collect the required results.

A similar approach was followed to objectify legacy GUI-driven applications (GDAs) and integrate them with other GDAs, including Web applications, to form mega-applications [GBP02]. This requires the use of agent processes to be injected into the GDAs. These agents collect information on all GUI elements that are used by a GDA, monitor events that are generated, and trigger GUI input events. An agent presents an object (representing an objectified GDA GUI) to a controlling program. This program can then invoke methods on specific GDA GUI elements and replay GUI inputs with the support of the agent. This controlling program can integrate a number of GDA GUIs together, including Web-based GUIs.

2.3 Software Requirements and Process Model Recovery

Requirements recovery is the process of retrieving software functional and user requirements and/or software specifications from an existing software, its documentation, its stakeholders and its operation environment. Requirements recovery research is fairly scarce. Previous work in this area had explored a variety of methods that assume different input information and recover various different types of requirements. In LeNDI, we developed interest in requirements recovery research because part of LeNDI's role in CelLEST process is to discover the system services that may be candidates for reengineering and wrapping with a new front-end. Generating hypotheses about the system services, as they are used today by current system users, in the form of interaction patterns, is essentially a requirements-recovery activity. So, it was important to review the related literature. Despite the variety of interesting approaches that were used for requirements recovery, none of them used interaction traces as input. Hence, LeNDI needed a novel method for interaction pattern discovery from interaction traces as described in chapter 6.

In the REVERE project [REGS00] probabilistic natural language processing (NLP) methods were employed to recover software requirements from the available documentation, such as requirements specifications, operating manuals, user interview transcripts and data models. The method suffers from the well-known shortcomings of NLP and needs to be adapted (trained) to the various documentation styles, structures and notations, but provides rapid analysis for voluminous documentation.

Cohen [Coh94] used Inductive Logic Programming (ILP) to discover specifications from C code. The constructed specifications are in Datalog (Prolog with no function symbols). The software discovered two thirds of the specifications with about 60% accuracy, in a program containing over one million lines of source code. The system recovered declarative view specifications from relational database examples. Positive examples were obtained from program execution views, with background knowledge, consisting of the table relation. The technique uses inductive reasoning about the behavior of the code, rather than deductive reasoning of static code. Sufficient training data is required, otherwise results will contain numerous inconsistent specifications.

The AMBOLS project [LAQ99] aimed to recover requirements by employing semiotic methods and intensive interviews with the stakeholders to analyze and model the system behavior from various viewpoints. The intent is to document current uses for the purpose of redeveloping the application.

In [SP99], data reverse engineering was proposed as a means for business rules recovery from legacy information systems. Particularly, an approach for extracting constraint-type business rules from database applications was outlined, but without an implementation or experimental evaluation.

Di Lucca *et al* [DFD00] presented a method for recovering a use case model from threads of execution of object-oriented (OO) code. A thread is a sequence of method executions linked by the messages exchanged between objects. Threads are triggered by input events and terminated by output events. In this approach, developers identify statements that form input events and output events. A tool then automatically identifies the code corresponding to the potential uses cases. The tool produces a structured use case model including diagrams at various levels of abstraction, comprising actors, use cases, associations between actors and use cases, and relationships among use cases. The mapping between a given use case to its corresponding code supports developers in program understanding and maintenance impact analysis. The method targets OO systems, which makes it inapplicable to most legacy systems that were developed before the wide spread of the OO paradigm.

Similar to this research area, is the work on process model discovery, e.g. [AGL98]. The idea is to model existing known or unknown processes by mining workflow and other logs of these processes and retrieving or discovering the models. The aim is to gain better understanding of the existing process models and/or develop future ones. An example of this work is reverse engineering work processes in collaborative virtual environments [BS02]. The goal was to recover design models of virtual workspaces at micro (individual tasks), macro (processes) and collaboration (task sequences) levels, by mining the environment's data logs, e.g. threads of email messages posted on the bulletin board and actions performed by the collaborating team members.

The work presented above represents diverse directions in exploring and tackling the requirements recovery problem. Researchers explored different available inputs, e.g., existing documentation, human knowledge, code, data, threads of OO program runs and workflow logs. In this thesis, we introduce a new method for recovering the *de facto* functional requirements of legacy systems to support the CelLEST method for legacy CUI reengineering and Web-enabling. To do so, we employ another yet unexplored easy-to-collect input, which is records of the system-user dialog via the legacy CUI in the form of interaction traces. LeNDI applies data mining algorithms to these traces to recover the needed requirements in the form of interaction patterns. There are a number of potential uses of this promising approach beyond the forward engineering phase of CelLEST, as suggested in section 7.3.

2.4 Sequential Data Mining

Mining sequences of data for recurring patterns is a generic problem with instances in a range of domains. It was first introduced in [AS95] under the name "sequential pattern mining", inspired by applications in the retail industry. Given a set of customers and their sequences of transactions, the goal is to discover sequences of items (patterns) occurring in the transactions of the same customer.

In CelLEST, it is necessary to discover the frequent legacy CUI navigation sequences that represent multiple uses of the same legacy system service, from a system viewpoint, or repetitive executions of the same user task, from a user viewpoint. LeNDI mines the recorded traces of interaction with a legacy system for these interaction patterns. This problem, called interaction pattern mining problem, is different from "mining sequential patterns", but similar to the problem of "discovery of frequent episodes in event sequences" [MTV97]. In [MTV97], the discovered frequent episodes or patterns can have different types of ordering: full (serial episodes), none (parallel episodes) or partial and have to appear within a user-defined time window. The support of a pattern is measured as the percentage of windows containing it. Some algorithms were developed to tackle this problem, e.g. WINEPI and MINEPI [MTV97] and Seq-Ready&Go [BCB00], based on the famous data mining Apriori algorithm [AIS93, AS94]. Apriori was originally proposed to solve the problem of mining association rules between sets of items in large databases and then numerously extended to solve other data mining problems including mining of sequential patterns. The problem of mining interaction patterns differs than the formulation of [MTV97] in that it does not restrict the pattern length with a window length and permits a user-defined number of insertion errors to exist in the instances of the discovered patterns.

The CelLEST interaction pattern discovery problem is also similar to the problem of discovering patterns in DNA and protein sequences. There, the objective is to discover either probabilistic patterns or deterministic patterns with noise, e.g. flexible gaps, wild-cards (don't care characters) and/or ambiguous characters (which can be replaced by any character of a subset of the alphabet set, A) [BDVHHP00]. Because bio-sequential data is usually very large, an efficient search strategy is to discover short or less ambiguous patterns using exhaustive search, possibly with pruning. Then the patterns that have enough support are extended to form longer or more ambiguous patterns. This process continues until no more patterns can be discovered. Two elegant examples of this category are PRATT [Jon96] and TEIRESIAS [Flo99] algorithms. PRATT can discover patterns of the quite general PROSITE format [BB94], e.g. C-x(5)-G-x(2,4)-H-[BD], where B,C, D, G and H $\in A$, x(5) is a flexible gap of length 0 to 5, x(2,4) is a flexible gap of length 2 to 4, and [BD] is an ambiguous character that can be replaced by B or D. The original TEIRESIAS algorithm discovers $\langle L,W \rangle$ patterns with wild-cards only, where $L \leq W$. An $\langle L,W\rangle$ pattern has a constraint on its density, that is any of its sub-patterns containing exactly L non-wildcards items has length of at most W items. For example, CD..CH..E is a $\langle 3,5\rangle$ pattern, where '.' can be replaced by one item $\in A$. None of these two bio-pattern discovery algorithms mentioned above suits the task of mining interaction traces for interaction patterns.

None of the problem formulations described above matched the needs of LeNDI, especially in terms of the type of ambiguity or errors that they allow in the patterns discovered. LeNDI discovers patterns of user activity in the traces of interaction with a legacy CUI with insertion errors, i.e., whose instances may contain up to a user-defined number of spurious activities. A spurious activity happens when the user accesses or receives a screen that is not part of the task s/he is performing, e.g., an error or help screen. LeNDI treats such activities as insertion errors and allows up to a pre-set number of them to exist anywhere in a pattern instance. Two novel pattern mining algorithms were developed, specifically to solve the interaction pattern mining problem in LeNDI: Interaction Pattern Miner (IPM) and Interaction Pattern Miner 2 (IPM2). The first is a breadth first algorithm and the second is a depth first algorithm. Both algorithms require defining a criterion for pattern selection and use the idea of building longer patterns from shorter ones. Although designed for use in LeNDI, they can be applied to similar problems. In fact we used them to mine user web site navigation logs for interesting navigation patterns [ES03]. Both algorithms are described, compared and evaluated in chapter 6.

Chapter Three

CelLEST User Interface Reengineering

In chapter 2, various legacy CUI reengineering, reverse engineering and Web-enabling methods were introduced. It was shown that almost all CUI reengineering and reverse engineering methods rely on code analysis and understanding, except some methods that deal with command-line programs. This makes such methods inapplicable when the code is unavailable or unchangeable. It also makes them costly and risky when code is hard to comprehend and difficult to change. Nevertheless, in some cases, it is unavoidable to do UI reengineering via code analysis and change since the code will be migrated or for other reasons. This raises the need for new CUI reverse and forward engineering methods that do not use code in order to serve cases when code change is impossible or undesirable. This thesis proposes a novel lightweight CUI reverse engineering process that utilizes, as input, traces of interaction with legacy CUIs. It is a valuable method when code and platform migration is not necessary and lightweight CUI reengineering will be used.

Additionally, chapter 2 presented the different strategies of Web-enabling legacy systems. It showed the reasons of popularity and advantages of Web-enabling via presentation access. These reasons are non-invasiveness, low risk, lightweight and low cost. It also discussed the shortcomings of this method, which are labor intensiveness, manual processes, inadequacy of tool support, vulnerability to unexpected events and to changes in the legacy CUI and inability to extend the system functionality significantly. The interaction reverse engineering method proposed in this work and implemented in LeNDI overcomes some of these shortcomings by proposing a coherent lightweight reverse engineering method for legacy CUIs in service of Web-enabling via presentation access. Unlike current manual industrial practices, this method is almost-automated, less error-prone, more productive and less sensitive to changes.

Since, a significant part of this thesis was done within CelLEST project for legacy CUI reengineering, it is important to describe this project first before describing the specifics of our interaction reverse engineering method. CelLEST project [SEKSM99, SES02, SESP00] is a joint research project between the Software Engineering Lab. at University of Alberta, Canada and an industrial sponsor, Celcorp [Cel]. The aim of the CelLEST project is to develop the next generation of legacy CUI reengineering and Web-enabling via presentation access, using artificial intelligence (AI) and other methods. CelLEST adopts interaction reengineering as a means to automate the process of "learning" and reengineering an existing legacy CUI. CelLEST uses a combination of document analysis, feature extraction, clustering, user action modeling, visualization, data mining, task model inference, XML wrapping and automated GUI layout to develop an intelligent semi-automated lightweight method and prototypes for legacy system CUI reengineering, Web-enabling and front-end integration. The CelLEST CUI reengineering is a two-phase process; the first is a reverse engineering phase and the second is a forward engineering phase. The overall CelLEST process is shown in Figure 3.1.

In the reverse engineering phase, the users' interaction with the legacy system CUI is recorded using a specially instrumented emulator. The recorded traces consist of the screen snapshots accessed by the users while navigating the legacy CUI, the actions they performed on these screen snapshots and the sequences they followed during their navigation to accomplish their work. Then, these traces are used to build a behavioral state-transition model for the legacy CUI (Task T1). This model is a road map for the legacy CUI. It is used by the new reengineered UI to verify the identity of legacy screen snapshots while they are accessed to perform a user task, and hence input the appropriate inputs and deduce the required outputs. Additionally, data mining algorithms are applied to the interaction traces to discover frequent patterns of interaction with the legacy system (Task T2). Each pattern is enriched with additional semantic information to build a model of the corresponding system service or user task, in terms of the interface navigation and the information exchange it implies (Task T3).

In the forward engineering phase of CelLEST, the user task models are translated in to abstract GUI specifications in XML (Task T4). These specifications are then translated to XHTML for Web access or WML (Wireless Markup Language) for WAP (Wireless Application Protocol) access, using the appropriate CelLEST interpreter (Task T5). The strength of this approach is in that it can accomplish simultaneous reengineering of the same legacy CUI to suit different platforms, using the abstract GUI specifications, which are platform independent.

The CelLEST method for semi-automated "learning", modeling, reengineering and Web-enabling of legacy CUIs and the user tasks of interest makes it much easier to deal

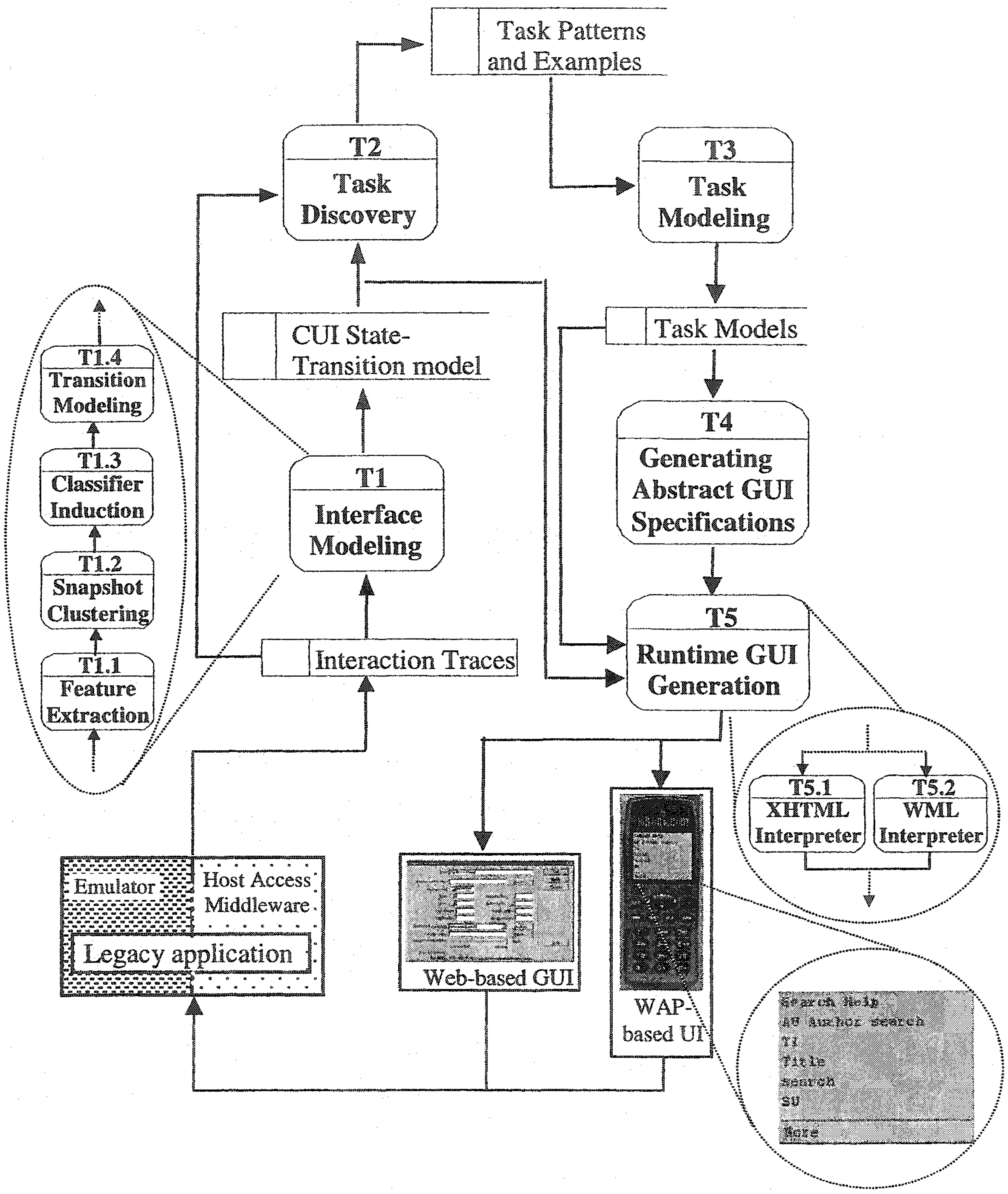


Figure 3.1. CelLEST User Interface Reengineering Process.

with frequently changing legacy systems, since manual re-modeling and GUI re-development may be quite costly. The following sections brief the CelLEST process and show its advantages over the current practices. In Figure 3.1, Tasks T1, T2 and T3 are the reverse engineering phase of the process. T1 and T2 were the subject of this work and are described in full details in this thesis. T4 and T5 are the forward engineering phase, and specifically T5 is a runtime task. T3, T4 and T5 were developed by other CelLEST project members and their different versions are described in detail in [Kap01, Kon00].

3.1 Interaction Traces Collection

While the users of the legacy system are performing their regular tasks, their interaction (dialog) with the legacy UI is recorded in the form of traces or sequences, using a specially instrumented emulator. For block-mode data transfer protocols like IBM 3270, a trace is a collection of screen snapshots forwarded by the legacy application to the user's terminal, interleaved with user actions in the form of sequences of keystrokes performed in response to receiving a screen snapshot. Additionally captured information, in case of IBM 3270 data streams, include the total number of fields, the number of unprotected fields and the initial cursor position. We call these recorded traces "interaction traces". Formally, an interaction trace is defined as follows:

Definition 3.1

$Trace_{id,n} = snap_{id,1}\ (key_{id,j}\ snap_{id,j})^*\ j = 2 \ldots n$, where

- *id* is the trace Id,
- *n* is the length of the trace,
- *j* is the j^{th} screen snapshot received at the user's terminal, and
- key_j is the sequence of key-presses issued by the user at snapshot $snap_{j-1}$ that caused the application to send the next snapshot $snap_j$ to the user's terminal.

3.2 T1: Legacy Interface Behavior Modeling.

The purpose of this task is to build a behavioral model for the legacy CUI, in the form of a state-transition model [EISSM01, SEIS03, SEKSM99]. Each node (state) of the model corresponds to a screen of the legacy system CUI, identified by a unique predicate. Using automatically extracted features for every screen snapshot recorded in the traces, a clustering algorithm groups similar snapshot together, as instances of the same screen. Then a classifier induction algorithm is applied to the snapshots of the identified clusters to automatically infer a unique predicate for their screen, i.e., for the corresponding node or state on the state-transition graph. Using a pattern-inference algorithm, the arcs of the state-transition model are inferred automatically. Each arc models a permissible user action. The details of the algorithms used in this task are given in chapters 4 and 5. The produced state-transition model is road map for the legacy CUI. It allows a new reengineered UI to check the identity of incoming snapshots while accessing them online against the nodes or states of the model, and hence input the appropriate inputs and extract the required outputs relevant to the executed user task. Additionally, the state-transition model can be queried about navigation paths from a state to another, thus helping planning new tasks that are achievable through the CUI although not originally intended by system developers. The LeNDI (*Legacy Navigation Domain Identifier*) prototype was developed to test the methods and algorithms used in this task. LeNDI deals with data transfer protocols that are native block-mode protocols or can be emulated in block-mode, e.g., IBM 3270 and VT100. In the sequel, we brief the subtasks of task T1 in Figure 3.1.

3.2.1 T1.1: Feature Extraction

In order to automatically build a legacy screen classifier that is able to distinguish the snapshots of each screen using unique screen signatures or predicates, one needs a rich set of features. LeNDI employs a variety of document analysis techniques to extract visual and other features for every snapshot. The output of this subtask is a feature vector for every snapshot. These features include:

- The existence of special system keywords, sentences or information at the top or bottom of the snapshot, e.g., title, code, date, time or page number.
- The information received with the outbound legacy data streams, e.g., the location and type (protected or unprotected) of IBM 3270 data fields and the cursor position.
- Snapshot layout features like the classification of the snapshot to “general”, “table” or “list” with some attributes for the last two classifications, e.g., the number of columns, rows, etc. Another layout feature is vertical and horizontal histograms built for the entire snapshot content or only for some special characters of interest, e.g., numbers.

LeNDI employs 14 single-part and multi-part features. Additionally, it has a binary[1] feature suite of 39 features, constructed by decomposing and abstracting these 14 features. This binary feature suite is needed for one of LeNDI's clustering algorithms. Chapter 4 describes in detail LeNDI's feature suite and all the algorithms used to extract them, along with the similarity measuring metric used for each feature.

3.2.2 T1.2: Snapshot Clustering

Snapshot clustering is the process of grouping similar snapshots together to infer their common identity, represented by a signature or a predicate that uniquely distinguishes them from other snapshots. LeNDI employs two clustering techniques. The first is a single-path incremental clustering algorithm [SEKSM99]. The second is a top-down clustering algorithm [EISSM01]. The incremental algorithm goes over the snapshot set only once, accessing it one by one. Using a user-defined similarity function, the algorithm assigns a new snapshot to the most similar cluster of the clusters available so far, or assigns it in a new cluster if it is not similar enough to any existing cluster. The algorithm requires the user to set up a similarity function and the similarity threshold that decides if a new snapshot is to be placed in a new cluster or to join an existing one. In addition, it requires sufficient familiarity with the system in hand. In return, it does not need an estimate of the number of clusters sought. The top-down algorithm uses LeNDI's binary feature set. It requires as input an estimate of the number of clusters and does not assume familiarity with the legacy system under analysis. Initially, it assigns all the snapshots in a single cluster and then keeps splitting clusters iteratively until reaching the desired number. In each iteration, the algorithm employs an internal cluster incoherence measure to split the most incoherent cluster using the feature value that minimizes the maximum incoherence of the clusters resulting from the split. This algorithm produces a decision tree that reflects the hierarchy of the splitting decisions used to produce the resulting clusters.

The user can choose which algorithm to use depending on the system under analysis. LeNDI's clustering process is interactive. The LeNDI analyst performs few rounds of clustering with different setups to enhance the obtained results. When reaching a satisfactory clustering of the data set, s/he can review and correct the results by moving

[1] Binary here means a feature whose comparison yields only either one or zero.

misclustered snapshots to their right clusters and joining redundant clusters with their originals.

3.2.3 T1.3: Classifier Induction

Given the snapshot clusters resulting from subtask T1.2, a classifier can be induced that can correctly classify individual snapshots into their corresponding clusters. This classifier can then be used at runtime to recognize new, previously unseen snapshots as instances of the CUI screens, and hence, to infer what actions are possible on each snapshot and to which screens they lead. In addition, verifying the snapshot identity allows the new reengineered GUI to apply whatever relevant input or output steps of a task plan to the snapshot, via the host-access middleware.

LeNDI employs two classifier induction algorithms. The first is a signature-based classifier that is induced by superimposing the snapshots of each cluster and capturing what is common in their feature vectors and presentation spaces. The second is a decision tree classifier, which is associated with the top-down clustering algorithm. It is induced by applying the user feedback for fixing the results of the top-down clustering algorithm to the decision tree produced by the algorithm. The fixed decision tree classifier is then used at runtime to infer the identity of new snapshots. The classifier produced should be used to classify new data to test its ability to generalize its knowledge and its accuracy.

3.2.4 T1.4: Transition Modeling

Transition modeling is the process of inferring a model for the transition needed to transfer the legacy system CUI from a screen to another, i.e., from a state on the state-transition model to another. Such a model includes the origin screen Id, the destination screen Id and a model of the user action needed to do the transition. Different styles of user-interaction with legacy systems exist, e.g., function keys, menu-driven, command-driven, and form-filling. Also, an action can have several formats; e.g., a command keyword may have multiple synonyms or it may have an equivalent function key. Currently, LeNDI can model command-driven and function keys styles. LeNDI infers each action model by comparing the instances of this action recorded in the interaction traces and applying a set of rules for command language design. For each action, LeNDI infers its syntax in terms of the function or control key(s) used and the command keyword(s), its options and its arguments. For the arguments, it infers their number, their syntax if any and whether they are optional or mandatory. The more instances are available, the more general and accurate the action model is. The LeNDI analyst can override and rewrite or fix an inferred action model.

3.3 T2 and T3: Frequent User Task Discovery and Modeling

The purpose of these two tasks is to automate the process of modeling the frequent user tasks of interest as much as possible. Hence, T2 and T3 save the labor-intensive work needed to define all the possible navigation paths of every task that needs to be reengineered and every piece of data exchange that takes place during the task. This is done by automatic learning from the interaction traces about the frequent user tasks in terms of what navigational path is traversed and what type of input is entered on which location on which screen for every task. To know what output is of interest to the user during a task, i.e., what information on which screen is retrieved, the analyst and/or an expert user need to manually identify this information on the snapshots of some instances of this task. This is because these outputs are visually retrieved by the legacy system user, i.e.; s/he just reads them on the screen or prints them. However, s/he does not take any action that can be recorded in the traces as evidences of which areas on the screen display these outputs.

3.3.1 T2: Task Discovery

LeNDI automates the discovery of users' frequent interaction patterns with the legacy system, which represent frequent uses of the legacy system services or frequent executions of the important user tasks. Two algorithms for sequential pattern mining were developed in LeNDI for this purpose, called Interaction Pattern Miner (IPM) [ESS02b] and Interaction Pattern Miner 2 (IPM2) [ESS02c]. Both algorithms can discover similar segments of interaction with the legacy system, in the recorded traces, even with some noise in the form of spurious irrelevant screens. Accommodating noise gives LeNDI flexibility in discovering tasks that include user mistakes, unnecessary navigation like invoking help screens and/or alternative paths for some subtasks. IPM is a depth-first algorithm, while IPM2 is a breadth-first algorithm. They require defining a criterion for interesting patterns in order to use it for deciding if a pattern is worthy of reporting or not. The criterion includes the pattern's minimum length, minimum number of occurrences, minimum score and the maximum number of insertion errors allowed in any instance of this pattern. The scoring function used is explained in chapter 6.

After reviewing the discovered patterns, the analyst needs to decide whether each one of them corresponds fully or partially to a real frequent user task, or is just a spurious repetition of a navigational path. The instances of each user task can then be used to build the corresponding task model.

3.3.2 T3: Task Modeling

Mathaino [Kap01, KS01, SK02] is another prototype tool of CelLEST. It accomplishes a reverse engineering task, T3, and two forward engineering tasks, T4 and T5. Mathaino replaced an earlier CelLEST tool URGenT (User interface ReGENeration Tool) [KSM99, SEKSM99, Kon00]. Mathaino generalized some of the concepts developed in URGenT by allowing more flexibility in defining task models and by supporting simultaneous legacy CUI migration to multiple platforms using intermediate platform-independent GUI representations, as opposed to the migration only to Java platform supported by URGenT. In T3, Mathaino analyzes the instances of each user task comparatively to construct an abstract model of:

- The navigational sequence through the legacy system UI to perform the user task;
- The types of information input by the user to the legacy UI and displayed to him/her through his/her navigation, and the locations where they occur on the legacy screens;
- The domain of values of the inputs; and
- The interdependencies among these values.

Note that to analyze the instances of each user task, evidences of the user inputs and outputs is necessary. All user inputs are already recorded in the interaction traces. The CelLEST process needs a user or an analyst to highlight on the snapshots of the task instances the areas that contain the outputs extracted to successfully complete the task.

Given the annotated task instances, Mathaino analyzes the flow of information to and from the legacy system to identify the user inputs required to accomplish the task, by studying all the recorded instances of this task. It compares the values used for each input field across all the task instances, and the values of all input and output fields in the same task instance. Each data input field is labeled with one of the following:

- **constant**, whose value is the same in all task instances,
- **derived**, whose value is obtained earlier in the task from an output field,
- **redundant**, whose value is input multiple times in the same task,
- **enumerated**, whose value is always one of a limited set of values, and
- **unpredictable**, whose value is independently supplied by the user.

Categorizing input fields leads to a significant reduction in the user input required by the reengineered UI of the task, e.g., the user will not need to input a derived input as it will be supplied automatically. Categorization helps choosing the proper abstract widget type for each input field in task T4, e.g., an input labeled with "range" can be implemented using a combo box or a set of radio buttons. CelLEST engineer may inspect the identified pieces of information and name them with meaningful names

Additionally, by comparative analysis of all instances of the same data field, Mathaino infers the coordinates of this field on the legacy screen it belongs to, in case these coordinates are static, i.e., the data filed always appears in the same location of the screen. In dynamic screens, such as free forms, attempts are made to discover starting and/or terminating landmarks to use for locating the data field.

Finally, if alternative paths exist for a user task or subtask, the branching screens need to be manually identified. Then, each alternative path is analyzed as described above. At runtime, the signature (predicate) of the snapshot received after performing an action on an instance of the branching screen decides which path to follow.

The task model produced in T3 specifies the path on the interface state-transition model through which the user navigates, the flow of information between the legacy application and the user, and the syntax of the interactions through which the information is exchanged. Effectively, it constitutes a declarative and executable specification of the modeled task of the legacy application. Given values for all the "unpredictable" pieces of information identified, the model can be used to drive the legacy application and execute the modeled task.

3.4 T4: Generating Abstract GUI Specifications

Mathaino uses model-based UI design heuristics to produce automatically abstract specifications for the new reengineered UI of each task, using its model. Thus, it eliminates the need for the current manual practice of piece-by-piece mapping of the task model to a GUI design. The specifications are described in terms of a set of abstract forms, each corresponding to a set of screens of the legacy system. Mathaino ensures that all the output fields identified in the task model are displayed on one of the forms. For each abstract form, a corresponding plan for navigating through the legacy screens at runtime is produced.

Using various heuristics, an abstract widget is proposed for each input or output field. For example, a field with an enumerated range of values is represented by a combo box or a set of radio buttons depending on the number of values and an "unpredictable" variable is represented by a text box. Then, the widgets are laid out on the form in a tabular manner. The user can override the default choices of widgets and layout settings. For example, s/he can change the widget type issued for a field or the number of layout columns. After applying user feedback, an XML representation of the abstract specifications is produced.

3.5 T5: Runtime GUI Generation

CelLEST runtime environment consists of two components. The front-end one is the runtime interpreter. It is responsible for interpreting the XML abstract GUI forms on a specific platform. It supplies widgets in the target platform that most closely match the abstract input widgets. Currently, an XHTML interpreter (for Web-enabling) and a WML interpreter (for WAP-enabling) are available [KS01].

The back-end component is the host navigator. It is built over an open source host-access middleware [JM00]. The host navigator executes the navigation plans of the abstract forms and passes the inputs to the legacy system and collects back the outputs. But first, the XHTML or WML interpreter passes the plan details to the host navigator in a platform-independent format.

3.5.1 T5.1: The XHTML Interpreter

For Web-enabling, the XHTML interpreter dynamically parses the XML abstract GUI forms at runtime and translates them to XHTML CGI forms. It maps the abstract GUI widgets to the appropriate CGI widgets. It uses XHTML tables to layout the produced web page in the closest format to that chosen by the user for the abstract GUI. Also, it parses the CGI response produced by the client Web browser into the platform-independent form needed by the back-end host navigator. It is a server-side component that runs as a Java servlet on the web server.

3.5.2 T5.2: The WML Interpreter

WML was developed by the WAP forum [WAP] for rendering web pages on WAP-enabled mobile Internet devices like Cellular phones and Personal Digital Assistants. A web page in WML (also called WML deck) is limited to a maximum of 1200 bytes. To overcome the device display limitation, a deck can be divided into a number of cards. The device can display only one card at a time. The only input widgets supported by WMP are simple text boxes. WML does not support CGI but provides some features that can simulate CGI.

The WML interpreter is adjusted to deal with these constraints. For example, it implements an abstract GUI form using several WML decks if 1200 bytes are not enough to implement the form. It uses a numerical menu to represent "enumerated" input fields. It internally caches the user responses to the multiple decks corresponding to a single abstract GUI form, before submitting it to the host navigator.

Chapter Four

Feature Extraction For Legacy Screen Snapshots

Building the state-transition model of the user interface of a legacy system requires identifying the states, i.e. the nodes, of this model. Each node represents a group of similar snapshots, instances of a distinct legacy interface screen, which corresponds to a behavioral state of the legacy system. For each legacy screen, it is necessary to identify a predicate that uniquely distinguishes the instances of this screen. To do so, using the current manual labor-intensive practices described earlier in chapter 2, one needs to do the following steps:

1. Study many snapshots of the screen of interest and sample snapshots of the other screens to discover how the former ones are similar to each other and different from the later, and
2. Find a predicate that uniquely distinguishes the snapshots of the screen of interest; this predicate can be a simple keyword or a complex textual pattern as described in subsection 2.2.3.

Task T1 of Figure 3.1 represents the process of building the state-transition model of a legacy CUI. LeNDI performs this task semi-automatically. Task T1 can be broken down to the following steps:

1. Extracting a rich set of features for every snapshot in the recorded traces, automatically,
2. Defining a similarity metric for each feature,
3. Defining a similarity and/or distance function to use for clustering similar snapshots together,
4. Clustering similar snapshots together, separate from the rest,
5. Verifying and correcting the clustering results via user feedback,
6. Automatically extracting unique predicates that distinguish the snapshots belonging to different clusters, i.e., to different legacy screens, and

7. Modeling the permissible user behavior (actions) on every legacy screen.

This chapter describes how the first two steps are implemented in this thesis. Steps 3 to 7 are explained in the chapter 5. This chapter presents the feature suite used in LeNDI, which resulted from detailed discussions with experts in the field of legacy CUI reengineering, analysis of many sample screen snapshots and experimentation. A combination of heuristics and document analysis methods is used to extract these features. We have tailored these methods to IBM 3270 data streams, a very popular block-mode data transfer protocol. In addition, we applied them to VT 100 emulated in block-mode. These features were developed to suit the automated state-transition building process, since the simple pattern-based features used in current practices, even those supported by the rich pattern languages used by some tools [BFM02, Cel99], are too simple for LeNDI's automated process. LeNDI has a base feature set used by its incremental clustering algorithm [SEKSM99]. Derived from the base set with some extensions, is a binary feature set used by LeNDI's top-down clustering algorithm. LeNDI's base feature set is divided into three subsets that cover different aspects of a snapshot. A snapshot consists of a presentation space, which is the matrix of characters displayed to the user on his/her terminal, when receiving the snapshot, and additional hidden information. The first feature subset includes features that are extracted from analyzing the periphery of the snapshot presentation space where important pieces of data are usually displayed, e.g., screen title, date, etc. The second subset includes features that are extracted from the IBM 3270 data stream hidden data that accompany the snapshot presentation space but are not visually displayed on the user terminal. The third subset includes features derived from analyzing the presentation space layout and content organization. The first and third subsets are generic features that can apply to any snapshots in block-mode data transfer protocol. The second subset is specific to IBM 3270.

Section 4.1 starts with a general discussion of the different types and styles of legacy screens that may be found in legacy systems. It gives a better understanding of the potential difficulties that may arise during snapshot clustering and the variety of features that would be needed to overcome them. Sections 4.2, 4.3 and 4.4 present LeNDI's three feature subsets. Each section starts by describing the intuition behind the feature subset it describes and follows by the detailed algorithms used to extract it. Then, it concludes by a description of the features of this subset and the similarity metric used with each feature along with example snapshots with the features extracted for them. Section 4.5 summarizes LeNDI's discrete feature set in an easy to reference tabular format. Section 4.6 presents the binary feature set, which is derived from the first set. Section 4.7 gives a description of LeNDI's feature extraction and viewing tool. Finally, section 4.8 is discussion and conclusions.

Before feature extraction starts for a snapshot, LeNDI evaluates whether or not its presentation space has the right dimensions for the legacy system under analysis which can be the same as the default of the data transfer protocol used or different. For example, IBM 3270 default presentation space dimensions are 24 rows x 80 columns. If the presentation space recorded by the recorder emulator of LeNDI is truncated, i.e., incomplete due to emulator or network error or whatever other reasons, LeNDI augments it to the matrix dimensions set in it so that feature extraction algorithms do not break. In the rest of this chapter, the topmost row of a snapshot is considered its first row (row number 1) and the leftmost column is considered its first column (column number 1).

4.1 Types of Legacy Screens

After studying samples of legacy screen snapshots, one can notice that different types of screens exist in terms of their content dynamics. Content dynamics is the variability of visual data fields that appear on the screen (not IBM 3270 data fields) in their number, contents, and locations. This directly influences the ease of clustering the instances of this screen together. Roughly speaking, one can recognize the following types of legacy screens, ordered from the most static to the most dynamic (See Figure 4.1).

1. Screens with a constant number of fields[2] displayed at fixed locations on the screen and containing static content. These are essentially "Static Screens". Examples of these screens are menus and system information and help screens. Such screens may have trivial variable items, e.g., the current date.

[2] The term field here refers to visual data fields as they appear on the screen. It does not refer to database fields or to IBM 3270 data fields.

2. Screens with a constant number of fields at fixed locations on the screen. Some of them have constant data content and the others have variable data content. Examples include input forms.

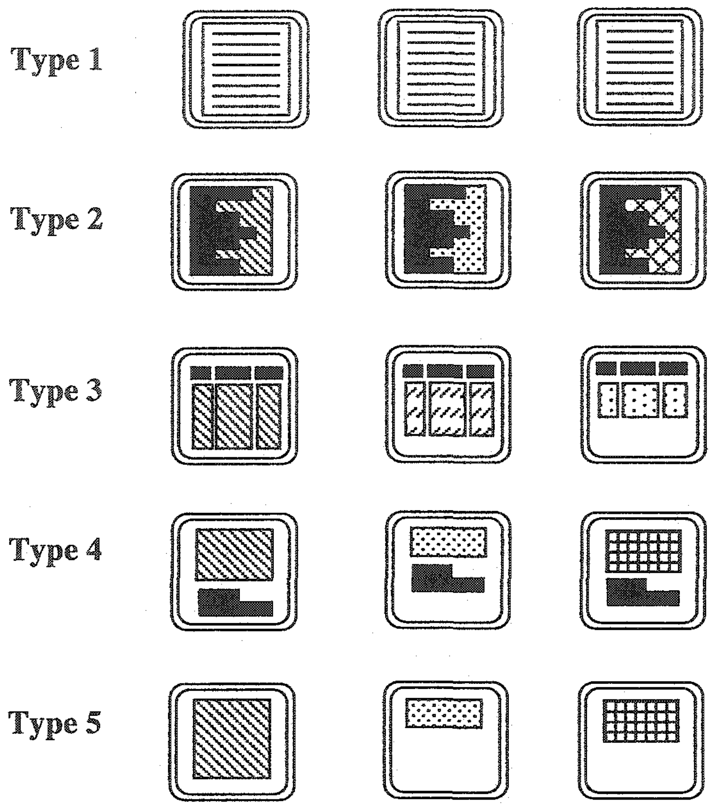


Figure 4.1. Different Types of Legacy Screens Ordered from The Most Static (upper) to The Most Dynamic (lower).

3. Screens with a constant part and a variable part in terms of the number of fields and their locations and contents. The constant part consists of a constant number of fields with constant contents and displayed at constant locations. The variable part consists of a variable number of fields with variable contents and displayed in a certain order, usually starting from a certain location. Examples include lists of information, e.g., lists of claims, employees, etc.
4. Screens with a constant part and a variable part, whose constant part consists of a constant number of fields with constant content but maybe displayed at different locations each time an instance of this screen appears. The variable part consists of a variable number of fields with variable content and displayed in a certain order, usually starting from certain location. Examples include screens of results of queries,

which may have a command line at the bottom prompting the user to enter another query. Note that if the constant part always appear at a certain location, e.g. the bottom of the screen, regardless of how filled the screen is, then, that screen will be of the type 3.

5. Screens with variable content. One can call them "Dynamic Screens". You can consider them as screens of the previous type without or with a trivial constant part that can be, e.g., a notice that a PF key returns the user to the previous screen. Examples of such screens are query result screens that retrieves textual data about a certain case, e.g., details of an insurance claim or a medical report about a patient. Usually the information starts from a certain location on the screen.

Generally speaking, clustering the snapshots of a screen together becomes harder as we go from the top to the bottom of the list of screen types above. But there are other factors that influence this process. In some cases, the existence of screen codes given by programmers, or clear titles, etc., makes it easier to group the snapshots together regardless of the nature of the rest of the content. In other cases, one screen may have different modes, e.g., Review and Update modes, which look almost alike, with few differences, mostly in the status of some of the data fields (protected, i.e., read only or input). The feature suite developed in LeNDI is broad enough to cover a variety of screen types. Therefore, it includes features based on special information in the periphery of the screen snapshot, e.g., code, title or date. It includes features based on the snapshot content and organization layout. It includes features that are not related to the snapshot visual appearance, but are rather based on the information received from the IBM 3270 data streams, e.g., IBM 3270 field information and the initial cursor position on the snapshot.

4.2 Presentation Space Features

Usually, some important information exists at the periphery of legacy CUI screens, e.g., title, screen code, date, etc. Discovering these pieces of information, their classifications and their locations on the snapshot presentation space is the base for this feature subset. The content and organization of this information can be very valuable in deciding the snapshot identity, and hence, in clustering it with similar snapshots. Of particular interest are screen titles or codes, which are often given to screens to make them easily recognizable by the legacy system users and developers.

4.2.1 Analysis of Presentation Space Periphery

LeNDI analyzes the first non-blank row, the row next to it and the last non-blank row of each screen snapshot to look for such information. LeNDI tries to discover the important pieces of information in these rows, if any. It assumes that reasonably big blank gaps in these rows are dividers or separators between such pieces of information. This assumption was examined and found to be true most of the time during our experiments with LeNDI. After extracting whatever pieces of information can be found in these rows, LeNDI classifies each of them to be screen code, title, date, time, page number, message. Moreover, LeNDI extracts the actual text of screen codes and titles, if any. Additionally, LeNDI extracts the cursor label or prompt message that prompts the user to input some data or command on the snapshot.

If the absolute first or last row of the snapshot is blank, the algorithm keeps descending (in case of top) or ascending (in case of bottom) until the first non-blank row is reached. In case of the first non-blank row, LeNDI considers the second one next to it even if it is blank. From now on, these two lines are called "the first two rows". Also, the terms "white space" and "blank" are used interchangeably. Algorithm 4.1 searches for such rows and if they are found, then, they are analyzed to discover any significant information that they may contain.

In Algorithm 4.1, step 3 loops through the rows of the given presentation space until finding a non-blank row or till the bottom of the presentation space is reached. *Pres Space* [*Count*] is the *Count*th row in *Pres Space*. Step 4 returns the message "Blank Screen" and terminates the algorithm if the snapshot is all blank. Otherwise, step 5 reports the first non-blank row and if it is not the last row, then it also reports the second row, regardless of its content, i.e., if it is blank or non-blank. Steps 6 to 8 find and report the last non-blank row.

Algorithm 4.1: Searching for The First Two and The Last Rows of a Presentation Space

```
Input: A presentation space, Pres Space, of a snapshot
Output: The first two and the last non-blank rows.
Steps:
1. Total Rows = The number of rows in Pres Space
2. Count = 1
3. While (Count $\le$ Total Rows) && (Pres Space [Count] is blank) do Count ++
4. If (Count $\ge$ Total Rows) then Return message "Blank Screen"
5. Else
   • Report Pres Space [Count] as the first non-blank row
   • If (Count < Total Rows)
         then Report Pres Space [Count+1] as the second row
   • Else Return message "The Screen Has Only One Non-blank Row "

6. Count = Total Rows
7. While (Count > 0) && (Pres Space [Count] is blank) do Count --
8. Report Pres Space [Count] as the last non-blank row
```

Algorithm 4.1. Searching for The First Two and The Last Rows of a Presentation Space.

Next, each of the first two rows is divided into three areas, using the longest blank sequences in the row as dividers between these areas. For the first row, the left, middle and right areas are numbered 1 to 3, respectively. The second row is similarly divided to areas 4, 5 and 6. Since the last non-blank row usually contains less information than the first two rows, it is divided into left and right areas only, which are numbered 7 and 8. The division algorithm used for the first two rows is Algorithm 4.2.

The idea of Algorithm 4.2 is to divide the given row into three areas using the biggest blank gaps in the row, which are thought to be the logical dividers used by developers. Therefore, the algorithm looks for the biggest two white spaces in the row to divide it. Steps 1 and 2 get the number of leading and trailing (left and right) spaces in the row, if any. Steps 3 to 4 get the length and location of the largest white space inside the non-blank content of the given row, if any. Steps 5 and 6 do the same but for the second largest white space. Steps 7 and 8 check whether the left and/or right spaces are longer than the largest space inside the row. If both are longer, then step 9 considers the left and right areas of the row to be empty and all the content found is classified to be in the middle, unless the middle is empty too. Also, this step reports the middle area content if found and then terminates the algorithm with a returning message of which areas are empty. In step 10, if the left space is the biggest, then the left area is considered empty and the row content is divided into middle and right areas. The function Substring (*string*, *start*, *end*) returns a string that is a sequence of characters taken from the parameter *string* starting from location *start* to location *end* inclusive. Step11 is similar to step 10 but applies when the right space is the biggest. Finally, step 12 deals with the case when the three areas of a row are considered non-empty, and the largest and second largest spaces are used to divide the Row into left, middle and right areas, depending on which of them comes first, i.e., starts at a smaller column number.

Algorithm 4.2 is used to divide the first two rows to three areas. A simpler algorithm is used to divide the last non-blank row into left and right areas. Its idea is to start from the left boundary of the snapshot and move forward in case of the left area and to start from the right boundary and move backward in case the right area. As long as there is some non-blank content, it is considered part of the corresponding area until 2 consecutive blanks are encountered. If no content is found until the middle column of the snapshot, the algorithm declares the corresponding area empty.

Algorithm 4.2: Dividing a Row to Right, Middle and Left Areas

Input: A row, *aRow*, taken from the presentation space of the snapshot under analysis.
Output: Three strings, representing the left, middle and right areas of *aRow*
Steps:

```
1. Left Space          = The number of leading spaces in aRow
2. Right Space         = The number of trailing spaces in aRow
3. LSpStart            = The starting position of the largest space inside aRow
4. LSp                 = The length of the largest space inside aRow
5. 2nd LSpStart        = The starting position of the 2nd largest space inside aRow
6. 2nd LSp             = The length of the 2nd largest space inside aRow
7. If (Left Space > LSp) then Left Area Empty = TRUE
8. If (Right Space > LSp) then Right Area Empty = TRUE
9. If (Left Area Empty) && (Right Area Empty) then
       Middle Area = Trim leading and trailing spaces (aRow)
       If (Middle Area is Blank) then Return message "All three areas are Empty"
       Else Report Middle Area and Return message "Left and Right Areas are Empty"
10. Else If (Left Area Empty) then
       Middle Area  = Substring (aRow, Left Space + 1, LSpStart - 1)
       Right Area   = Substring (aRow, LSpStart + LSp, |aRow| - Right Space)
       Report Middle Area and Right Area
       Return message "Left Area is Empty"
11. Else If (Right Area Empty) then
       Left Area    = Substring (aRow, Left Space + 1, LSpStart - 1)
       Middle Area  = Substring (aRow, LSpStart + LSp, |aRow| - Right Space)
       Report Left Area and Middle Area
       Return message "Right Area is Empty"
12. Else
       If (2nd LSpStart > LSpStart) then
           Left Area   = Substring (aRow, Left Space + 1, LSpStart - 1)
           Middle Area = Substring (aRow, LSpStart + LSp, 2nd LSpStart - 1)
           Right Area  = Substring (aRow, 2nd LSpStart + 2nd LSp, |aRow| - Right Space)
       Else
           Left Area   = Substring (aRow, Left Space + 1, 2nd LSpStart-1)
           Middle Area = Substring (aRow, 2nd LSpStart + 2nd LSpStart, LSpStart - 1)
           Right Area  = Substring (aRow, LSpStart + LSp, |aRow| - Right Space)
       Report Left Area, Middle Area and Right Area
```

Algorithm 4.2. Dividing a Row to Right, Middle and Left Areas.

Classification	Keywords	Patterns
Date	Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, January, February, March, April, May, June, July, August, September, October, November, December, Jan, Feb, Mar, Apr, Jun, Jul, Aug, Sep, Oct, Nov, Dec,	!!/!!/!! !/!!/!! !!/!/!! !/!/!!
Time	A.M., P.M., AM, PM,	!!:!!:!! !!:!!
Page	Page	! Of !
Message	Message, Error, Command, Ready, Return to, Enter, Type, PFKey, Found	PF! PF ! F! F !

Table 4.1. The Default Keyword and Pattern Lists of LeNDI ("!" means any digit).

After extracting the areas 1 to 8, each area is classified into one of the following seven categories, which are represented by codes 0 to 6:

- 0 → the area is empty
- 1 → the area contains a screen code
- 2 → the area contains a screen title
- 3 → the area contains date information
- 4 → the area contains time information
- 5 → the area contains page number information
- 6 → the area contains a message

An empty area is a blank one, which was reported to be blank by Algorithm 4.2 or its simpler version, or belongs to a row that was reported to be all blank by Algorithm 4.1. A non-empty area is checked for the existence of any of the keywords and/or patterns that may help classifying it to date, time, page information or message. Table 4.1 shows the default keyword and pattern lists used for this classification. Using LeNDI, one can tailor these lists for individual legacy systems by adding new items or removing unwanted ones based on his/her judgment and analysis of the screen style of the system in hand. If a non-empty area does not contain date, time, page number or message information, then it is classified to contain code or title. It is assumed to be a code if it contains one word. If it contains more than one word, then it is assumed to be title. Algorithm 4.3 categorizes the content of a given area into one of the seven categories.

```
Algorithm 4.3: Area Classification
Input: A string, area, representing one of the key areas of a snapshot.
Output: A classification of this area.
Steps:
1. Break area into words, using white spaces as dividers. Store the words in Word List
2. If Word List is empty then Return message "Empty"
3. If Word List contains a date keyword or pattern then Return message "Date"
4. If Word List contains a time keyword or pattern then Return message "Time"
5. If Word List contains a page keyword or pattern then Return message "Page"
6. If Word List contains a message keyword or pattern then Return message "Message"
7. If |Word List| = 1 then Return message "Code"
8. Else Return message "Title"
```

Algorithm 4.3. Area Classification.

4.2.2 Five Presentation Space Features

Four features are derived out of the analysis of the snapshot important areas. A fifth feature is extracted from the cursor label. The sequel discusses them.

4.2.2.1 Feature 1-1: Eight Areas Encoding

This feature is an encoding of the classification of the eight extracted areas, 1 to 8. For each snapshot, the value of this feature is an eight characters string, e.g., "03520106". The similarity measure of two values of this feature is the number of matching characters divided by 8. A 0 (empty area) and a 6 (message area) are considered a match since it is common that a legacy CUI allocates an area for system messages, if any message is to be presented to the user, which is empty otherwise.

4.2.2.2 Feature 1-2: The Start Columns of Titles and Codes

This feature is a string that is formed by concatenating the starting column of all title and code areas discovered, ordered from area 1 to 8. For example, assume that area 2 is a title that starts at column 23 and area 8 is a code that starts at column 65. The other areas are classified as empty, date, time, page information or message. Then, Feature 1-2 for this snapshot is "2365". This feature is useful in identifying screens whose peripheries are static in terms of the contents and their starting column locations. Two values of this feature are compared using binary comparison whose outcome is either one if the two values are identical, or zero otherwise.

4.2.2.3 Features 1-3 and 1-4: Titles, Codes and/or Selected Text Areas

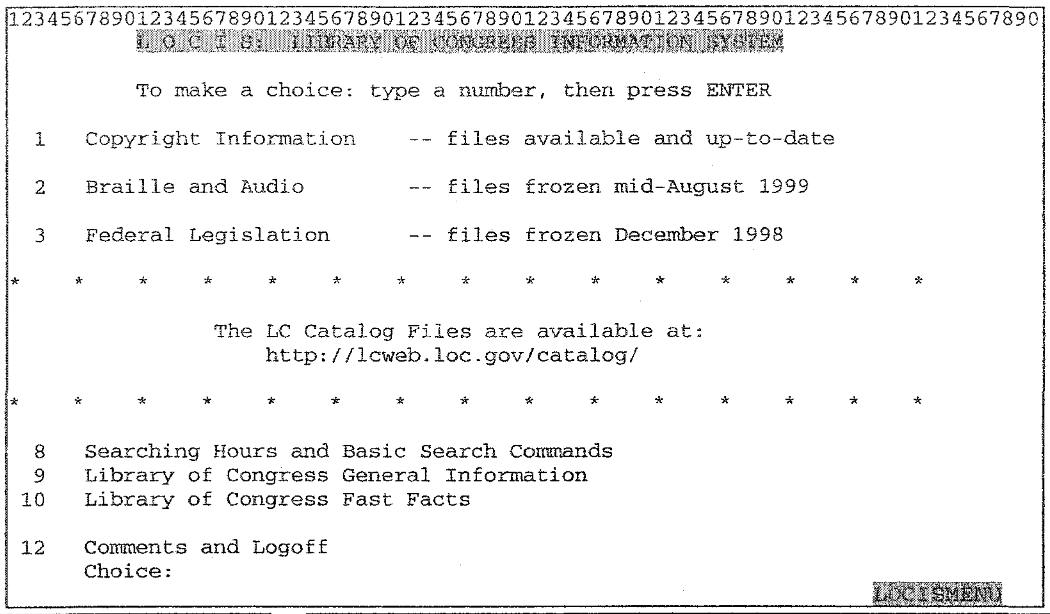
These two features are two of the eight areas extracted that are chosen for exact string comparison. This means that whatever in these areas will be recorded as such as a feature. However, numerical values are replaced by "!"s and spaces are removed to allow flexibility in comparison. For example, if the content of an area chosen for exact string comparison is "Items 1-3 of 13", the actual string value stored for this feature will be "Item!-!of!". The LeNDI analyst can open the feature extraction setup dialog box to choose two of the eight areas extracted for these two features. Alternatively, s/he can ask LeNDI to do so. LeNDI picks the two areas that are classified as codes or titles, over the entire snapshot set, more times than any other areas. Binary comparison is used to measure the similarity of two values of each of these features.

4.2.2.4 Feature 5-1: Cursor Label

LeNDI records the initial cursor position when a screen snapshot is received. The cursor's label is the last sentence or word up to 12 characters to the left of the cursor. This cursor label is Feature 5-1. Analysis of many snapshots of different legacy systems showed that the benefit of this feature depends on the style used to design the legacy CUI. Some systems have a standard command line or a few cursor labels shared among most legacy screens, making this feature less useful. In other cases, the variety of cursor labels and prompts increases the utility of this feature. Binary comparison is used to compare cursor label values.

4.2.3 Presentation Space Features Examples

This subsection includes a few snapshots, taken from a legacy system whose behavior was modeled using LeNDI. On each snapshot, each of the eight areas is marked with gray if it contains some text and is left blank if it is empty. Following each snapshot, is a table with LeNDI's classification of its eight key areas and the values of Features 1-1, 1-2, 1-3, 1-4 and 5-1. The examples are shown in Figures 4.2 to 4.7. The areas chosen for Features 1-3 and 1-4 are areas 1 and 2. If some area is classified as a code or title, then its start and end columns are included between brackets, e.g., (10-59). Note that the LeNDI starts columns' and rows' indices from 0, but in the figures below the indices start at 1 for ease of comprehension. Also, note that in Figure 4.6, the bottommost non-blank line left and right areas are the same, i.e., the same content is classified as the left and right areas. This is because the algorithm starts from the left and keeps moving right while no double space is encountered. It stops at the end of the message displayed. It does the same thing but starting from the right and moving backwards to get the right content, but ends up with the same message content. The keyword and patterns lists used for area classification are the default ones shown in Table 4.1.

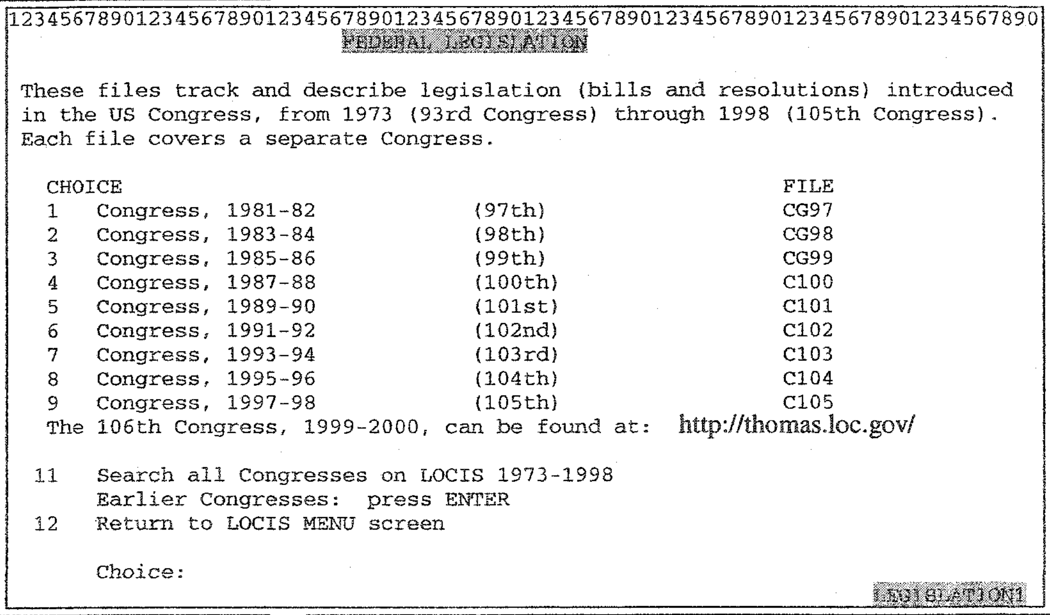


```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
          L O C I S:  LIBRARY OF CONGRESS INFORMATION SYSTEM

          To make a choice: type a number, then press ENTER

  1   Copyright Information    -- files available and up-to-date

  2   Braille and Audio        -- files frozen mid-August 1999

  3   Federal Legislation      -- files frozen December 1998

*    *    *    *    *    *    *    *    *    *    *    *    *    *    *

                The LC Catalog Files are available at:
                    http://lcweb.loc.gov/catalog/

*    *    *    *    *    *    *    *    *    *    *    *    *    *    *

  8   Searching Hours and Basic Search Commands
  9   Library of Congress General Information
 10   Library of Congress Fast Facts

 12   Comments and Logoff
      Choice:
                                                                   LOCISMENU
```

Area	Classification	Code
1	Empty	0
2	Title (10-59)	2
3	Empty	0
4	Empty	0
5	Empty	0
6	Empty	0
7	Empty	0
8	Code (67-75)	1

Feature 1-1	02000001
Feature 1-2	1067
Feature 1-3	BLANK
Feature 1-4	LOCIS:LIBRARYOFCONGRESSINFORMATIONSYS TEM
Feature 5-1	Choice:

Figure 4.2. An Example Legacy Screen Snapshot (1) with Features 1-1, 1-2, 1-3, 1-4 and 5-1 Extracted.

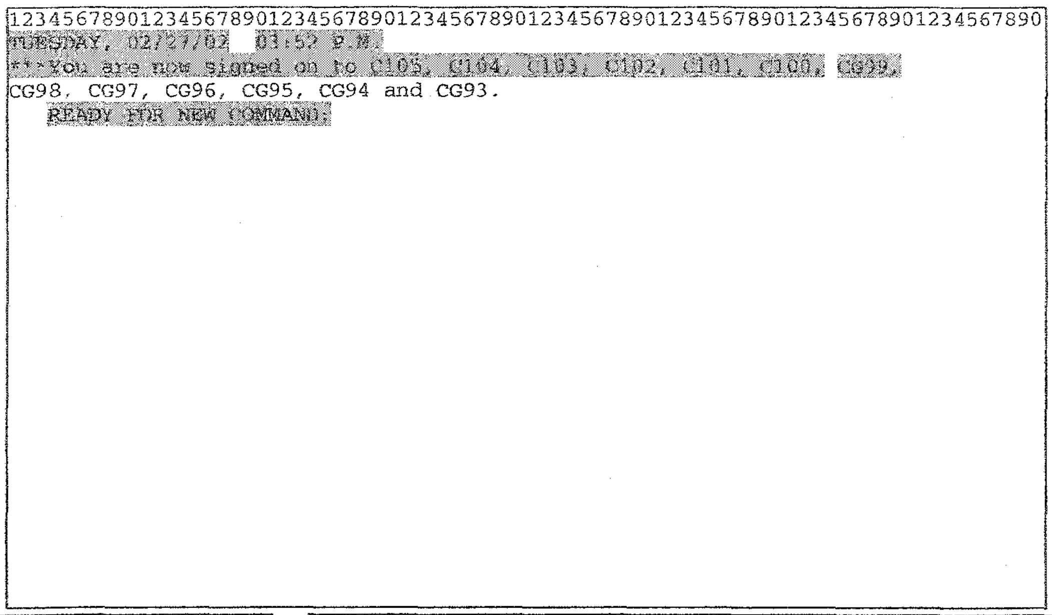


```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
                          FEDERAL LEGISLATION

 These files track and describe legislation (bills and resolutions) introduced
 in the US Congress, from 1973 (93rd Congress) through 1998 (105th Congress).
 Each file covers a separate Congress.

   CHOICE                                                    FILE
   1   Congress, 1981-82          (97th)                     CG97
   2   Congress, 1983-84          (98th)                     CG98
   3   Congress, 1985-86          (99th)                     CG99
   4   Congress, 1987-88          (100th)                    C100
   5   Congress, 1989-90          (101st)                    C101
   6   Congress, 1991-92          (102nd)                    C102
   7   Congress, 1993-94          (103rd)                    C103
   8   Congress, 1995-96          (104th)                    C104
   9   Congress, 1997-98          (105th)                    C105
   The 106th Congress, 1999-2000, can be found at:  http://thomas.loc.gov/

  11   Search all Congresses on LOCIS 1973-1998
       Earlier Congresses:  press ENTER
  12   Return to LOCIS MENU screen

       Choice:
                                                                LEGISLATION1
```

Area	Classification	Code
1	Empty	0
2	Title (26-44)	2
3	Empty	0
4	Empty	0
5	Empty	0
6	Empty	0
7	Empty	0
8	Code (67-78)	1

Feature 1-1	02000001
Feature 1-2	2667
Feature 1-3	BLANK
Feature 1-4	FEDERALLEGISLATION
Feature 5-1	Choice:

Figure 4.3. An Example Legacy Screen Snapshot (2) with Features 1-1, 1-2, 1-3, 1-4 and 5-1 Extracted.

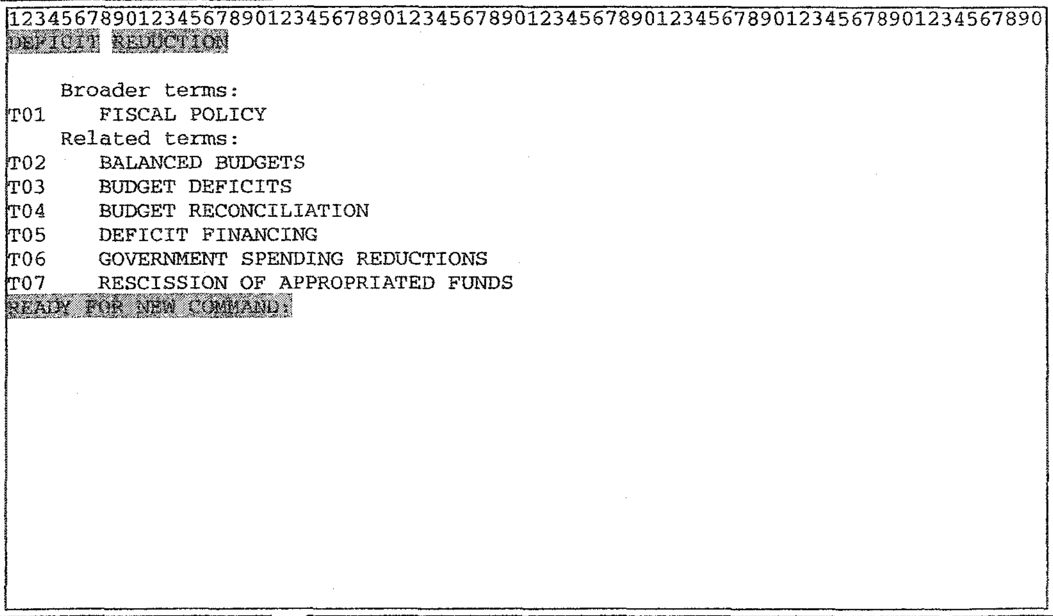


```
1234567890123456789012345678901234567890123456789012345678901234567890123456789O
TUESDAY, 02/27/02  03:52 P.M.
**You are now signed on to C105, C104, C103, C102, C101, C100, CG99,
CG98, CG97, CG96, CG95, CG94 and CG93.
   READY FOR NEW COMMAND:
```

Area	Classification	Code
1	Date	3
2	Time	4
3	Empty	0
4	Title (0-62)	2
5	Code (64-68)	1
6	Empty	0
7	Message	6
8	Empty	0

Feature 1-1	34021060
Feature 1-2	064
Feature 1-3	TUESDAY,!/!/!
Feature 1-4	!:!P.M.
Feature 5-1	NEW COMMAND:

Figure 4.4. An Example Legacy Screen Snapshot (3) with Features 1-1, 1-2, 1-3, 1-4 and 5-1 Extracted.

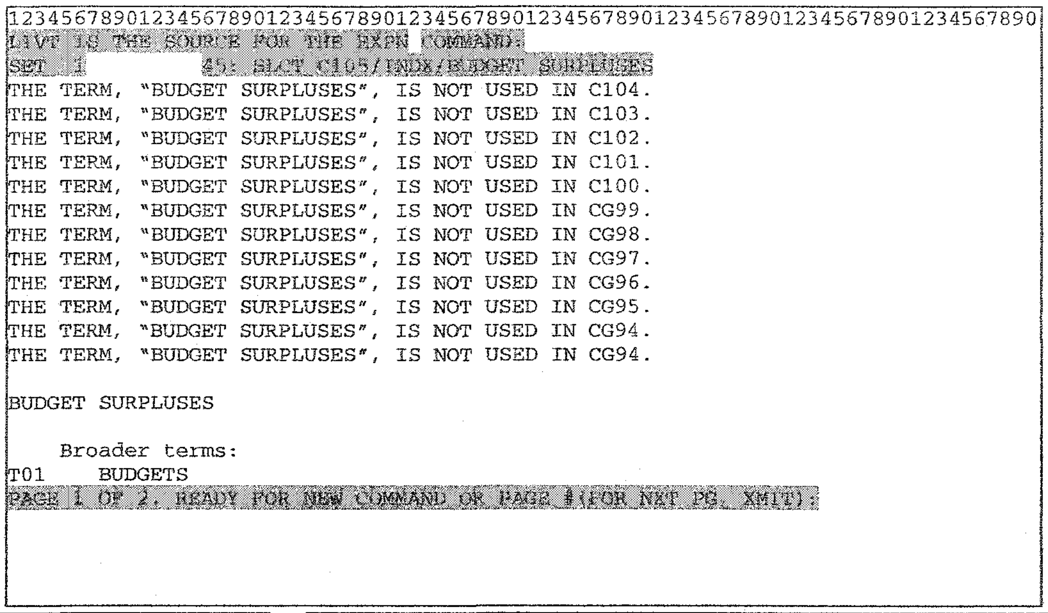


```
1234567890123456789012345678901234567890123456789012345678901234567890123456789O
DEFICIT REDUCTION

    Broader terms:
T01    FISCAL POLICY
    Related terms:
T02    BALANCED BUDGETS
T03    BUDGET DEFICITS
T04    BUDGET RECONCILIATION
T05    DEFICIT FINANCING
T06    GOVERNMENT SPENDING REDUCTIONS
T07    RESCISSION OF APPROPRIATED FUNDS
READY FOR NEW COMMAND:
```

Area	Classification	Code
1	Code (0,6)	1
2	Code (8,16)	1
3	Empty	0
4	Empty	0
5	Empty	0
6	Empty	0
7	Message	6
8	Empty	0

Feature 1-1	11000060
Feature 1-2	08
Feature 1-3	DEFICIT
Feature 1-4	REDUCTION
Feature 5-1	NEW COMMAND:

Figure 4.5. An Example Legacy Screen Snapshot (4) with Features 1-1, 1-2, 1-3, 1-4 and 5-1 Extracted.

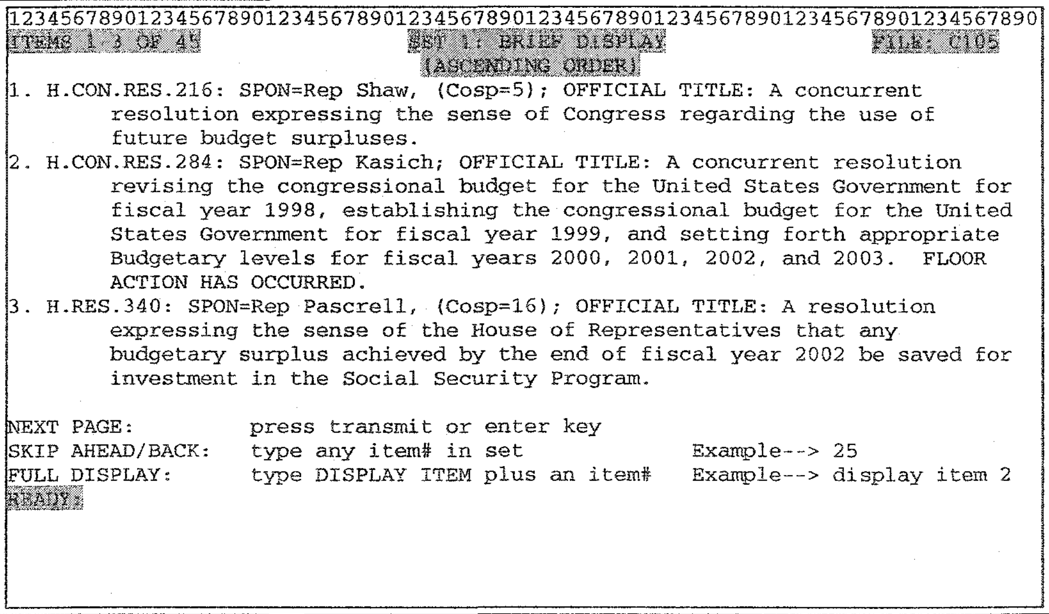


```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
LIVT IS THE SOURCE FOR THE EXPN COMMAND:
SET  1             45: SLCT C105/INDX/BUDGET SURPLUSES
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C104.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C103.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C102.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C101.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C100.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG99.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG98.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG97.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG96.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG95.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG94.
THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG94.

BUDGET SURPLUSES

    Broader terms:
T01    BUDGETS
PAGE 1 OF 2. READY FOR NEW COMMAND OR PAGE #(FOR NXT PG, XMIT):
```

Area	Classification	Code
1	Title (0,30)	2
2	Code (32,39)	1
3	Empty	0
4	Title (0-5)	2
5	Title (15-49)	2
6	Empty	0
7	Page	5
8	Page	5

Feature 1-1	21022055
Feature 1-2	032015
Feature 1-3	LIVTISTHESOURCEFORTHEEXPN
Feature 1-4	COMMAND:
Feature 5-1	XT PG, XMIT):

Figure 4.6. An Example Legacy Screen Snapshot (5) with Features 1-1, 1-2, 1-3, 1-4 and 5-1 Extracted.



```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
ITEMS 1-3 OF 45                  SET 1: BRIEF DISPLAY                  FILE: C105
                                  (ASCENDING ORDER)
1. H.CON.RES.216: SPON=Rep Shaw, (Cosp=5); OFFICIAL TITLE: A concurrent
        resolution expressing the sense of Congress regarding the use of
        future budget surpluses.
2. H.CON.RES.284: SPON=Rep Kasich; OFFICIAL TITLE: A concurrent resolution
        revising the congressional budget for the United States Government for
        fiscal year 1998, establishing the congressional budget for the United
        States Government for fiscal year 1999, and setting forth appropriate
        Budgetary levels for fiscal years 2000, 2001, 2002, and 2003.  FLOOR
        ACTION HAS OCCURRED.
3. H.RES.340: SPON=Rep Pascrell, (Cosp=16); OFFICIAL TITLE: A resolution
        expressing the sense of the House of Representatives that any
        budgetary surplus achieved by the end of fiscal year 2002 be saved for
        investment in the Social Security Program.

NEXT PAGE:          press transmit or enter key
SKIP AHEAD/BACK:    type any item# in set              Example--> 25
FULL DISPLAY:       type DISPLAY ITEM plus an item#    Example--> display item 2
READY:
```

Area	Classification	Code
1	Page Info	5
2	Title (31-50)	2
3	Title (67-76)	2
4	Empty	0
5	Title (32-48)	2
6	Empty	0
7	Code (0-5)	1
8	Empty	0

Feature 1-1	52202010
Feature 1-2	3167320
Feature 1-3	ITEMS!-!OF!
Feature 1-4	SET!:BRIEFDISPLAY
Feature 5-1	READY:

Figure 4.7. An Example Legacy Screen Snapshot (6) with Features 1-1, 1-2, 1-3, 1-4 and 5-1 Extracted.

4.3 IBM 3270 Data Stream Features

When receiving a snapshot via the IBM 3270 outbound data stream, LeNDI gets the presentation space of the snapshot along with some non-visual information. From this extra information, LeNDI records the initial cursor location on the screen and IBM 3270 field locations, lengths, attributes and protection status (protected or unprotected, i.e., read only or read/write). In some systems, this information can be very useful in clustering similar snapshots together. In particular, these features can help distinguish the snapshots of visually similar CUI states, e.g., those belonging to different modes of a multi-mode screen, if each mode is to be treated as a separate screen.

This non-visual extra information varies between data transfer protocols. The discussion below and the features in this subset apply to IBM 3270 data transfer protocol. For IBM 3270, LeNDI extracts two features that encode important information about the IBM 3270 fields received with the outbound data stream.

4.3.1 Feature 2-1: Hashing of the Number and Locations of IBM 3270 Fields

The first feature is a hash function of the information of the IBM 3270 data fields retrieved with a screen snapshot. It encodes two pieces of information, the number of fields and their locations. LeNDI uses the following hashing function:

$$\text{Feature 2-1} = \sum_{i \in I} (x_i \times y_i) + 10000 \times n \qquad I \text{ is the set of IBM 3270 data fields}$$

where x and y are the horizontal and vertical locations of the field's first character on the snapshot as received from the IBM 3270 outbound data stream and recorded by LeNDI and n is the number of 3270 fields on the current screen snapshot. This feature is numerical and its values are compared with one another using binary comparison.

4.3.2 Feature 2-2: The Number of IBM 3270 Unprotected Fields

This feature is the number of unprotected (input) data fields received from the outbound data stream carrying the screen snapshot. So, after LeNDI records all the 3270 data fields received, it counts the number of unprotected ones. This feature is more useful in clustering snapshots in systems with intense data entry operations, than other systems. Binary comparison is used to compare different values of this feature.

4.4 Presentation Space Layout Features

Looking at a screen snapshot as document, one can see that many screens have some layout structure that serves presenting the data on the screen in a meaningful comprehensible format. For example, some screens present their content as an itemized list, table, form, etc. Even when no clear structure is imposed on the screen, often its content is organized using a particular layout. For example, some columns or rows may be always denser in content than others on the snapshots of the same screen. Or, some numerical contents always exist at certain parts of the screen. In some other cases, some characters like '|', '-' or '*' are used to impose some patterns on the screen, e.g., vertical or horizontal dividers or frames. The features of this subset capture such layout characteristics. They are extracted using a number of image processing and document analysis methods. They are grouped in two groups: projection profiles features and layout classification features. The first group includes features that reflect the distribution of the entire content or special types of contents (e.g., numbers) on the rows and columns of the screen snapshot. The second group includes features that classify the layout to "table", "list" or "general" and the specifications of this classification, if it is one of the first two.

4.4.1 Projection Profiles

The features included in this subset are derived from different projection profiles built for every recorded snapshot. Projection profiles analysis is used for document understanding and mainly for separating different document components [SLGSH92, LHHP96]. Projection refers to the mapping of a two-dimensional region of an image into a waveform whose values are the sums of the values of the image points along some specified direction. A projection profile is obtained by determining the number of black pixels that fall into a projection axis. If the vertical and horizontal axes are chosen, then the corresponding vertical and horizontal projection profiles are histograms representing the number of black pixels in the columns and rows of pixels of a document image, respectively. Projection profiles represent a global feature of a document and play an important role in document component extraction. A deep valley in the profile with a certain predefined width is called a *cut*. Analysis of these cuts helps in separating the components of a document. Further details can be found in [SLGSH92, LHHP96]. LeNDI treats a screen snapshot as a document and considers every character as a "black pixel". Then, it applies projection profiles analysis to infer some features that describe the density and distribution of the snapshot content.

LeNDI builds five types of profiles for every snapshot; each is the base for one feature. The first two profiles are the vertical and horizontal binary histograms of the entire snapshot content. A binary histogram is the one that has one bit for every column or row represented. So, instead of recording the exact number of pixels per row or column, a "1" or "0" is recorded depending on whether the number of pixels is above a given threshold or not. The third is a vertical binary histogram of the numerical content of the snapshot. In building this histogram, non-numerical characters are treated as blanks. The forth is a histogram of the number of words in the two top and two bottom lines. The fifth is a binary histogram of a single character of a group of characters of interest that are thought to be used to impose some patterns on the screen snapshot, e.g., "-" or "_". LeNDI chooses the most frequent character on the given snapshot from the group of interesting characters for this fifth histogram.

Several measures for histogram distances are suggested in [SLGSH92]. To compare legacy snapshots projection profiles, LeNDI uses different versions of the normalized Euclidean similarity measure, which is the number of matching bits divided by the total number of bits. The rest of this section describes in detail the five types of projection profiles used in LeNDI and then presents the features associated with them and the similarity measures used for each feature.

4.4.1.1 All Characters Binary Vertical Profile

This is a binary encoding of the histogram produced by projecting all the snapshot content along the vertical axis. First, two setup parameters are retrieved from the database. These are the "Upper Vertical Cut" and the "Lower Vertical Cut". They define how many rows to cut off the presentation space from the top and the bottom, respectively, before building this profile. The default value for each of these parameters is 3, but The LeNDI analyst can override the defaults. The reason for these cuts is that legacy screens usually have date, time, title, list of available commands and other information at their tops and/or bottoms. This information may have a common layout among many screens but does not reflect any special layout characteristics of the individual screens, especially if projected vertically. This information at the periphery of the screen is used for extracting the presentations space features of section 4.2.

Second, the number of non-blank characters is counted per column. Then, for a certain column, if this number is above a certain threshold[3], it is represented by "1" in the binary encoding of the vertical profile; otherwise it is represented by "0". Otherwise, LeNDI uses the default value of 3. Setting this threshold gives The LeNDI analyst freedom to eliminate the noise produced by scattered characters and focuses the histogram representation on the body of the snapshot. The resulting histogram is 80 bits long for default IBM 3270 screen snapshots. It is stored as a string in hexadecimal format. Algorithm 4.4 is used for building this profile.

4.4.1.2 All Characters Binary Horizontal Profile

This is a binary encoding of the histogram produced by projecting the snapshot content along the horizontal axis. It is similar to all characters binary vertical profile, except that non-blank characters are counted per row not per column. No cuts are made to the snapshots before building the profile. A user-defined or a default horizontal threshold

---

Algorithm 4.4: Constructing All Characters Binary Vertical Profile
Input: A presentation space, *Pres Space*, of a snapshot
Output: The "all characters binary vertical profile" of *Pres Space*
Steps:

```
1. Retrieve the Upper Vertical Cut and Lower Vertical Cut from the database.
2. Retrieve the Vertical Threshold from the database.

3. Cut the top Upper Vertical Cut rows from Pres Space
4. Cut the bottom Lower Vertical Cut rows from Pres Space

5. Create String all char vertical profile

6. For every column in Pres Space
     • Count = the number of non-blank characters in this column
     • If (Count > Vertical Threshold) then Concatenate '1' to all char vertical profile
     • Else Concatenate '0' to all char vertical profile
7. Convert all char vertical profile to hexadecimal representation
8. Report all char vertical profile
```

Algorithm 4.4. All Characters Binary Vertical Profile Construction Algorithm

[3] LeNDI provides a default threshold which can be overridden by the analyst.

```
Algorithm 4.5: Constructing Numbers Binary Vertical Profile
Input: A presentation space, Pres Space, of a snapshot
Output: The "numbers binary vertical profile" of Pres Space
Steps:
1. Retrieve the Numbers Vertical Threshold from the database.
2. Create String numbers vertical profile
3. For every column in Pres Space
   • Count = the number of digits in column
   • If (Count > Numbers Vertical Threshold)
         then Concatenate '1' to numbers vertical profile
   • Else Concatenate '0' to numbers vertical profile
4. Convert numbers vertical profile to hexadecimal representation
5. Report numbers vertical profile
```

Algorithm 4.5. Numbers Binary Vertical Profile Construction Algorithm

is used to decide whether the count of non-blank characters in a row should be represented by "1" or "0".

4.4.1.3 Numbers Binary Vertical Profile

This profile is similar to the "all characters binary vertical profile" except that only the number of digits per columns is counted and all other characters are treated as blanks. No upper or lower cuts are made to the snapshot. The count of digits per column is compared to a user-defined or a default value of the "Numbers Vertical Threshold" setup parameter, to decide whether to represent it by "1" or "0" in the profile. Algorithm 4.5 is used to build this profile.

4.4.1.4 Words Horizontal Profile

This profile is a histogram of the number of words in the absolute top two rows and the absolute bottom two rows of a snapshot. A *word* is a horizontal sequence of characters that is preceded and succeeded by at least one space or by the left or right boundary of the screen snapshot. This definition includes line segments, numbers, etc. This is the only non-binary profile, meaning that the actual count of words is stored in hexadecimal format not just a binary encoding of it.

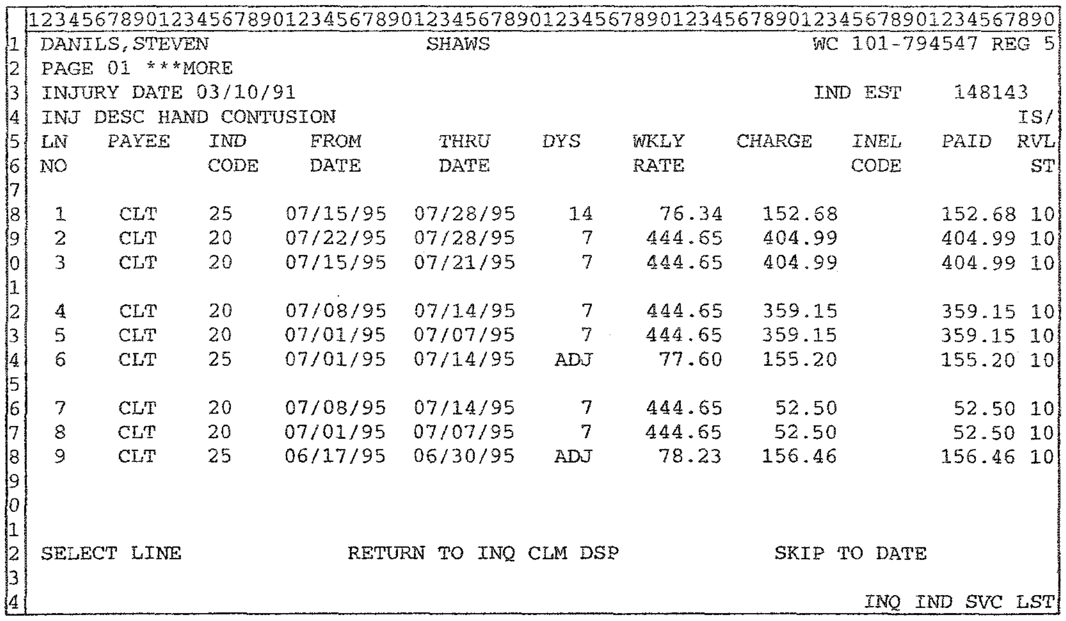
4.4.1.5 Special Characters Binary Profile

Some characters like "_", "|" or "*" are often used to create patterns, e.g., dividers, etc. on legacy screens. Some other characters like ":" or "." may be used in some "consistent" way on the snapshots of one screen, forcing some pattern on the instances of this screen. Consistency can be relative to the snapshot rows or columns, e.g., if a table column appears starting at the same column on a screen and it contains real numbers, then a pattern of dots ('.') will be formed on this snapshot. See an example of two snapshots of the same screen in Figures 4.8 and 4.9. One can notice that the patterns imposed by '/' and '.' on these two snapshots are more consistent in the vertical direction than the horizontal one, i.e., the same columns in both snapshots contain instances of these characters, but only some rows do. A binary profile that is created by capturing the consistent presence (i.e. pattern) of the most frequent of these characters on a snapshot can serve as a feature for comparing snapshot similarity.

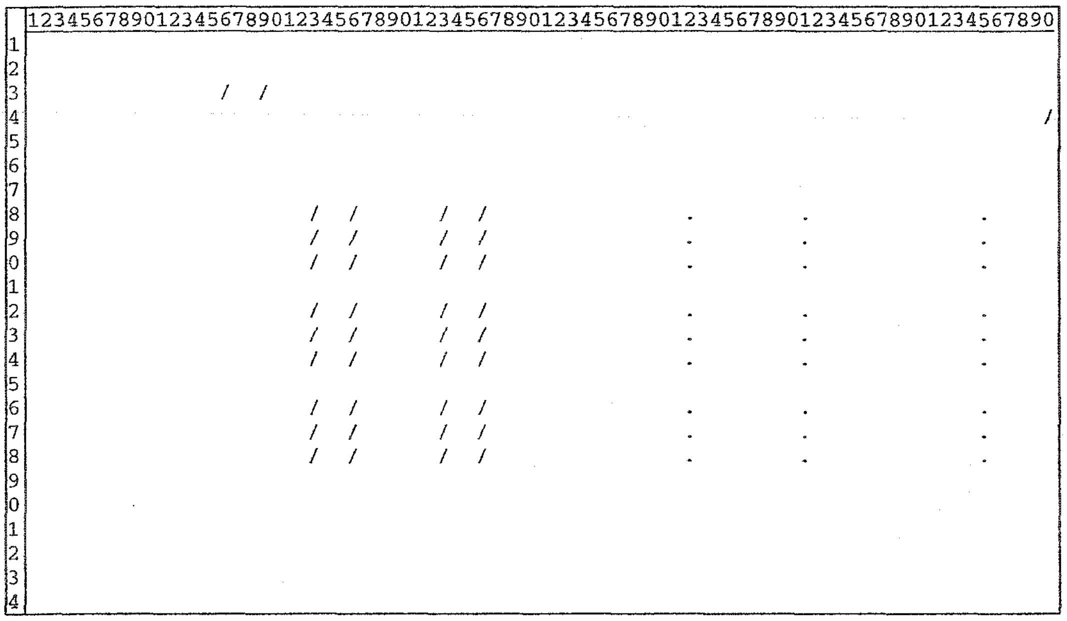
To implement this idea, first, LeNDI offers a default set of special characters and for each character, it offers a suggested direction (horizontal or vertical) along which, the corresponding character is thought to exist consistently more that the other direction. Table 4.2 shows this set. For example, "|" is usually used to create vertical lines or dividers on snapshots. So, it is suggested to build a vertical binary profile for it, if it is chosen as the character of interest. The LeNDI analyst can change or replace the default special characters set. LeNDI can accept up to 10 special characters. Additionally, s/he can change the type of profile suggested for a special character. Third, during feature extraction for a snapshot, LeNDI counts the number of occurrences for each of the special characters in the snapshot presentation space. For the most frequent character, it builds the corresponding type of profile associated with this character and encodes it in binary format. The upper and lower cuts used in building the "all characters binary vertical profile" are used in building the special characters profile too if it is a vertical one. If it is a horizontal one, no cuts are made.

Special Character	#	:	*	/	-	_
Profile Type	V	V	V	V	H	H

Table 4.2. LeNDI's Default Special Characters Set.



```
 DANILS,STEVEN                      SHAWS                                 WC 101-794547 REG 5
 PAGE 01 ***MORE
 INJURY DATE 03/10/91                                                     IND EST    148143
 INJ DESC HAND CONTUSION                                                                  IS/
 LN   PAYEE    IND      FROM        THRU       DYS     WKLY      CHARGE     INEL      PAID  RVL
 NO            CODE     DATE        DATE               RATE                 CODE             ST

  1    CLT     25     07/15/95   07/28/95       14      76.34    152.68              152.68 10
  2    CLT     20     07/22/95   07/28/95        7     444.65    404.99              404.99 10
  3    CLT     20     07/15/95   07/21/95        7     444.65    404.99              404.99 10

  4    CLT     20     07/08/95   07/14/95        7     444.65    359.15              359.15 10
  5    CLT     20     07/01/95   07/07/95        7     444.65    359.15              359.15 10
  6    CLT     25     07/01/95   07/14/95      ADJ      77.60    155.20              155.20 10

  7    CLT     20     07/08/95   07/14/95        7     444.65     52.50               52.50 10
  8    CLT     20     07/01/95   07/07/95        7     444.65     52.50               52.50 10
  9    CLT     25     06/17/95   06/30/95      ADJ      78.23    156.46              156.46 10



 SELECT LINE                RETURN TO INQ CLM DSP             SKIP TO DATE

                                                                            INQ IND SVC LST
```

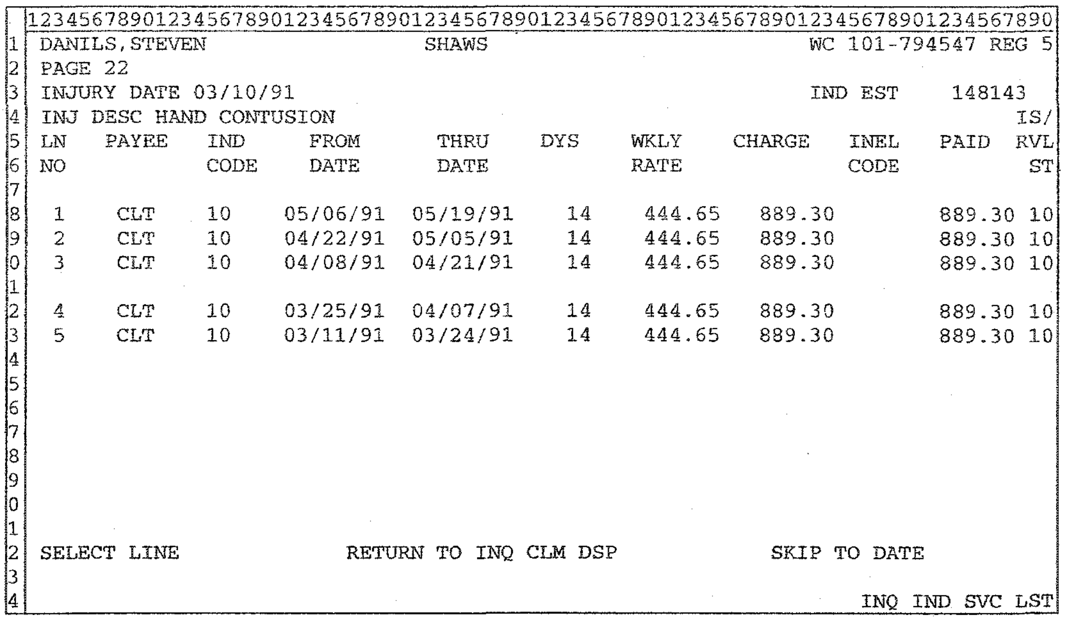


```


                 /  /
                                                                                          /



                        /  /        /  /                  .         .               .
                        /  /        /  /                  .         .               .
                        /  /        /  /                  .         .               .

                        /  /        /  /                  .         .               .
                        /  /        /  /                  .         .               .
                        /  /        /  /                  .         .               .

                        /  /        /  /                  .         .               .
                        /  /        /  /                  .         .               .
                        /  /        /  /                  .         .               .
```

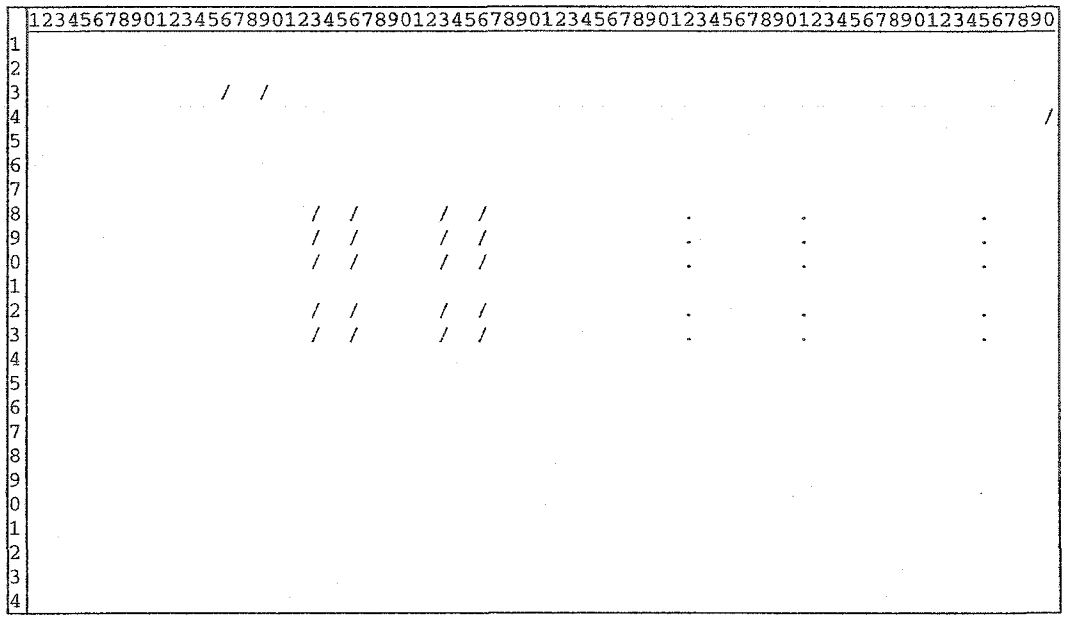
Figure 4.8. An Example Legacy Screen Snapshot (7) (upper). The Patterns Imposed on The Snapshot by '/' and '.' Characters (lower).

4.4.1.6 Features 6-1 and 6-2: All Characters Binary Vertical and Horizontal Profiles

Five features are derived for each snapshot from the projections profiles described above. The first is Feature 6-1. It is the "all characters binary vertical profile" of the snapshot. It is stored as a string of 20 characters; each represents a hexadecimal digit that encodes 4 bits, i.e., 4 columns of the profile. Feature 6-2 is similar to Feature 6-1. It is a 6-characters string encoding the "all characters binary horizontal profile". LeNDI does discrete comparison for 2 values of any of the Features 6-1 and 6-2, using the normalized



```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
 DANILS,STEVEN                 SHAWS                         WC 101-794547 REG 5
 PAGE 22
 INJURY DATE 03/10/91                                        IND EST    148143
 INJ DESC HAND CONTUSION                                                     IS/
 LN   PAYEE   IND     FROM      THRU     DYS    WKLY    CHARGE   INEL   PAID  RVL
 NO           CODE    DATE      DATE            RATE             CODE          ST

  1    CLT    10    05/06/91  05/19/91    14    444.65   889.30        889.30 10
  2    CLT    10    04/22/91  05/05/91    14    444.65   889.30        889.30 10
  3    CLT    10    04/08/91  04/21/91    14    444.65   889.30        889.30 10

  4    CLT    10    03/25/91  04/07/91    14    444.65   889.30        889.30 10
  5    CLT    10    03/11/91  03/24/91    14    444.65   889.30        889.30 10








 SELECT LINE             RETURN TO INQ CLM DSP            SKIP TO DATE

                                                                 INQ IND SVC LST
```



```
12345678901234567890123456789012345678901234567890123456789012345678901234567890


               /  /
                                                                               /



                      /  /      /  /               .        .             .
                      /  /      /  /               .        .             .
                      /  /      /  /               .        .             .

                      /  /      /  /               .        .             .
                      /  /      /  /               .        .             .
```

Figure 4.9. An Example Legacy Screen Snapshot (8) (upper). The Patterns Imposed on The Snapshot by '/' and '.' Characters (lower).

Euclidean similarity measure, which is the number of matching bits divided by the total number of bits. The resulting value represents how similar two snapshots are, based on this feature. In other words, this value shows how similar the contents of both snapshots are distributed across snapshot columns (Feature 6.1) or rows (Feature 6.2).

4.4.1.7 Features 6-3: Numbers Binary Vertical Profile

Feature 6-3 is a 20-character string that encodes the "numbers binary vertical profile" in a hexadecimal format. A weighted Euclidean similarity measure is used to compare values of this feature. This means that the weight of matching "1"s can be different than that of matching "0"s. The LeNDI analyst may define (or use the default) weights to decide how important the coexistence of a "1" in the same bit on both values compared to the coexistence of a "0". The formal case indicates that the corresponding columns on both snapshots include some numerical content, while the later indicates that the columns do not have any numerical content. The weight of a mismatch is 0.

4.4.1.8 Features 6-4: Words Horizontal Profile

Feature 6-4 is the "words horizontal profile". It is a string formed by concatenating the 4 numbers representing the word counts for the absolute two top and absolute two bottom lines of the snapshot. The counts are in hexadecimal format. The similarity of two values is the number of matching counts (out of 4) divided by 4.

4.4.1.9 Features 6-5: Special Characters Binary Profile

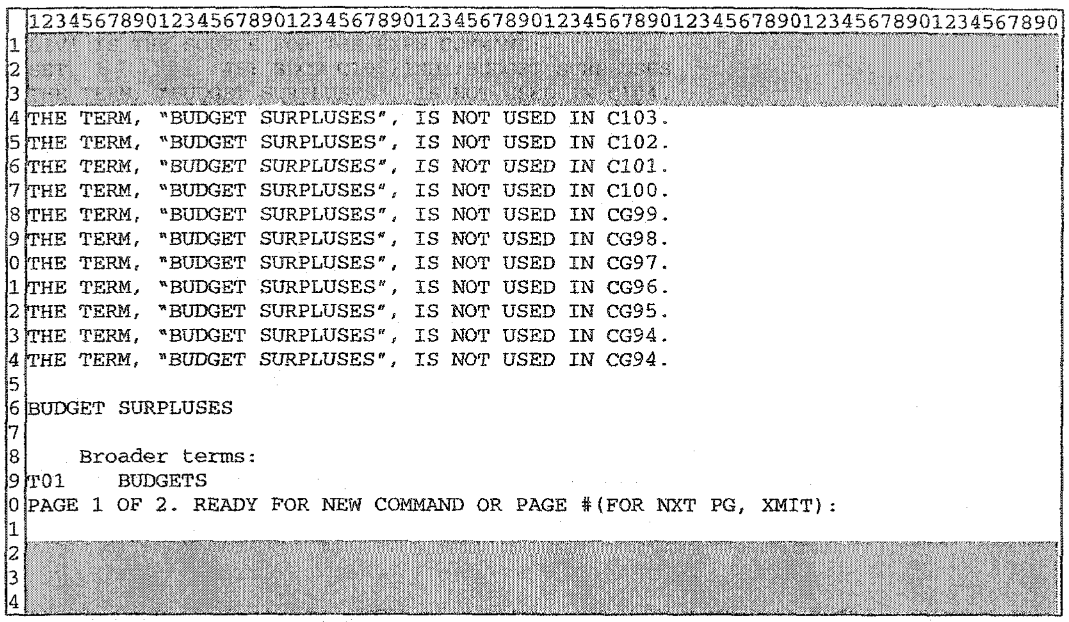
Finally, Feature 6-5 is an encoding of the "special characters binary profile". It is a 7-characters or 21-chracters string depending on whether the profile is horizontal or vertical, respectively. In both cases, the first character of the string is an encoding of which special character is used. For example, if the special characters set contains 6 characters, then the first of them is given the code "0". The second is "1", etc. When comparing two values of this profile, LeNDI starts by comparing the first character, to see if the same special character was considered for both snapshots or not. In case of a mismatch, LeNDI stops and the comparison result is 0. In case of a match, LeNDI proceeds to compare the entire profile using the weighted Euclidean similarity measure used with Feature 6-3.

4.4.1.10 Projection Profiles Example

This subsection provides an example legacy screen snapshot and the five projection profiles produced for it and the associated features. Figure 4.10(a) shows the setup parameters used in this example. Figure 4.10(b) shows the snapshot used in the example and the upper and lower vertical cuts. Figure 4.10(c) to (g) show the five projection profiles produced for this snapshot and the values of the corresponding features. Note that the least significant digit in Features 6-1, 6-2, 6-3 and 6-5 represent the 4 left most columns or the 4 topmost rows, depending on the direction of the profile. So the profile "1110111110111111....." will be represented by the string "......fdf7". Additionally, note that Feature 6-5 in Figure 4.10(h) starts with the character '5' which indicates that the sixth special character was chosen for building the special character profile of the given snapshot. According to Figure 4.10(a), this character is ',' and the associated projection direction is vertical.

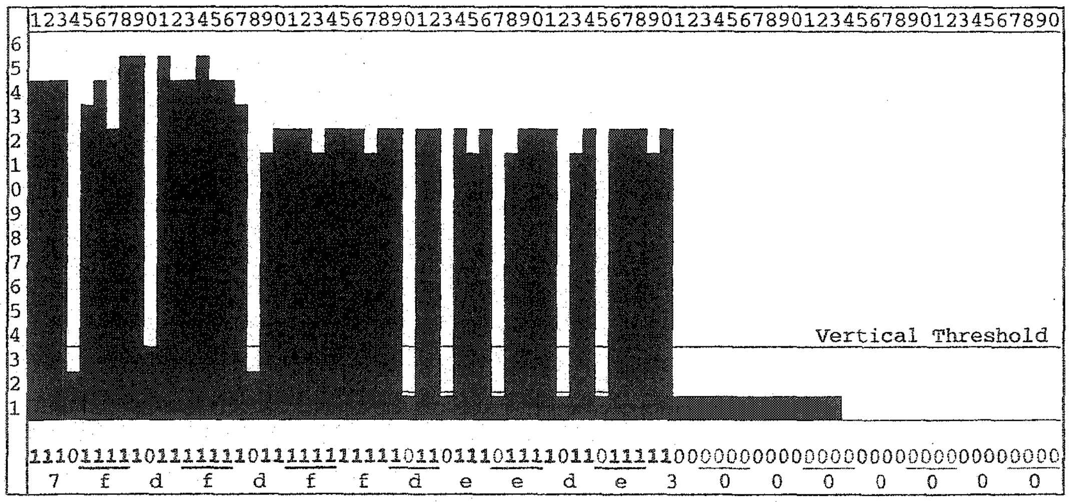
Upper Vertical Cut	3	Numbers Threshold	3
Lower Vertical Cut	3	Special Character Set	- (H) _(H) #(V) :(V) *(V) ,(V)
Vertical Threshold	3	1-1 Match Weight	1.5
Horizontal Threshold	10	0-0 Match Weight	0.3

(a) The setup parameters used in this example. #(H) means if the special character '#' is chosen, build a horizontal profile for it.



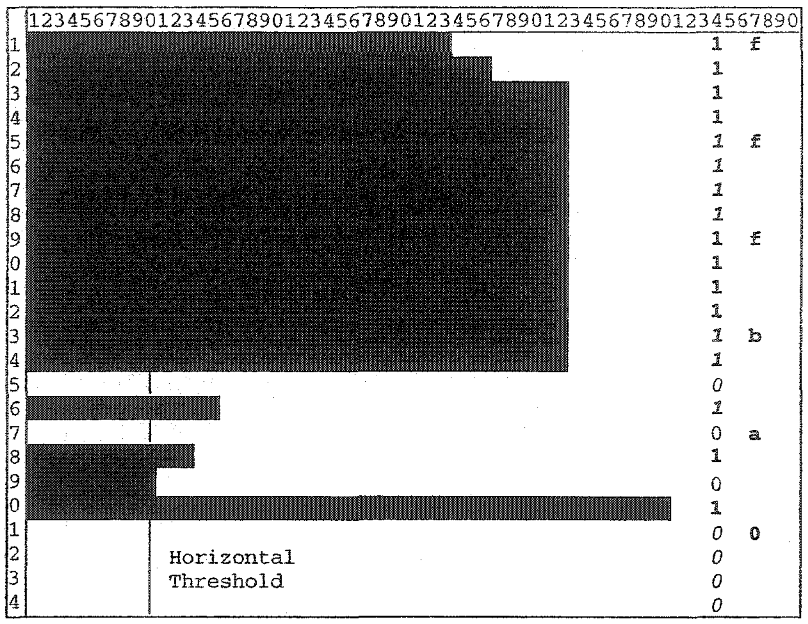
```
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
1[illegible]
2[illegible]
3[illegible]
4THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C103.
5THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C102.
6THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C101.
7THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C100.
8THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG99.
9THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG98.
0THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG97.
1THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG96.
2THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG95.
3THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG94.
4THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG94.
5
6BUDGET SURPLUSES
7
8    Broader terms:
9T01     BUDGETS
0PAGE 1 OF 2. READY FOR NEW COMMAND OR PAGE #(FOR NXT PG, XMIT):
1
2
3
4
```

(b) An example legacy screen snapshot (9). Upper and lower cuts are in gray.

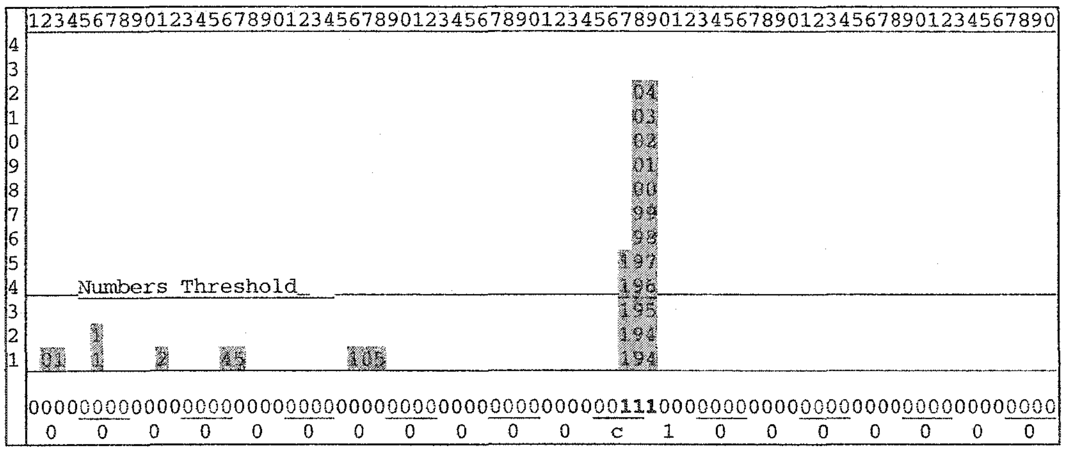


(c) A vertical projection of the content of the example snapshot in (b) and the corresponding "all characters binary vertical profile" at the bottom.

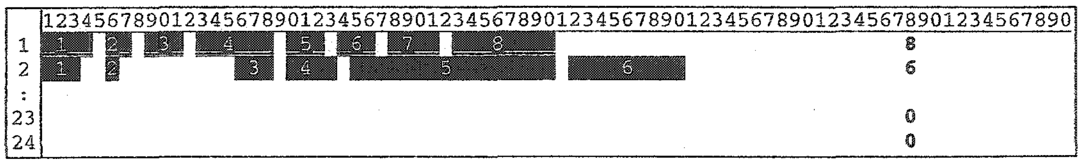
Figure 4.10(a-c). An Example Legacy Screen Snapshot (9) with Features 6-1, 6-2, 6-3, 6-4 and 6-5 Extracted.



(d) A horizontal projection of the content of the example snapshot in (b) and the corresponding "all characters binary horizontal profile" to the right.

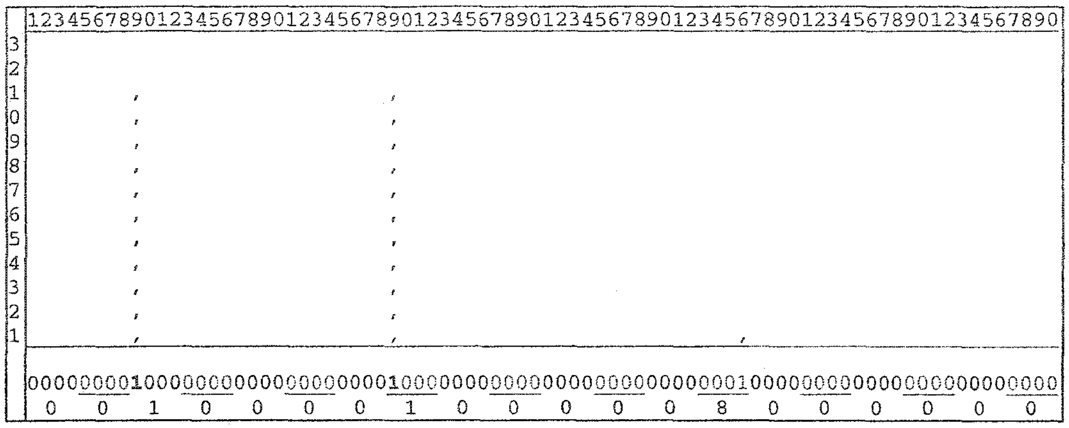


(e) A vertical projection of the numerical content of the example snapshot in (b) and the corresponding "numbers binary vertical profile" at the bottom



(f) The words horizontal profile of the snapshot in (b). The words of the top 2 and bottom 2 rows are numbered and the word count per column is to the right.

Figure 4.10(d-f). An Example Legacy Screen Snapshot (9) with Features 6-1, 6-2, 6-3, 6-4 and 6-5 Extracted.



```
12345678901234567890123456789012345678901234567890123456789012345678901234567890
00000000100000000000000000001000000000000000000000000001000000000000000000000000
 0   0   1   0   0   0   0   1   0   0   0   0   0   8   0   0   0   0   0   0
```

(g) The vertical projection of the special character ',' for the snapshot in (b) and the corresponding "special characters binary projection profile".

Feature 6-1	00000003edeedffdfdf7
Feature 6-2	0abfff
Feature 6-3	00000001c00000000000
Feature 6-4	8600
Feature 6-5	50000008000010000100

(h) Features 6-1 to 6-5 for the snapshot of (b).

Figure 4.10(g,h). An Example Legacy Screen Snapshot (9) with Features 6-1, 6-2, 6-3, 6-4 and 6-5 Extracted.

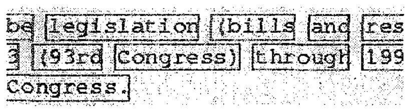
4.4.2 Layout Classification

It is very common for legacy screen snapshots to have some format, e.g., table, list, or other format. It would be very useful in grouping similar snapshots together to discover their formats and specifications, e.g., the number of column and rows of a table or the number of items in a list. Based on this, the second feature subset derived from snapshot layout is derived by analyzing the positions of the different components of the snapshot content relative to each other and deciding whether or not these positions impose a certain structure on the snapshot appearance. Currently, LeNDI can classify a snapshot to table, list or, general screen. Additionally, it discovers the specifications of the discovered format. To do this, a number of document analysis algorithms are applied to the snapshot, which is treated as a document. First, LeNDI tries to discover any tabular structure on the document. If it fails, then it tries to discover if a list exists on the snapshot. If it fails, then it labels the snapshot as "general", which means that no structure layout could be discovered. This classification is used as Feature 7-1. Feature 7-2 is a summary of the specifications of the table or list discovered, if any. The following details the algorithms used, and follows by feature description and examples.

4.4.2.1 Table Detection: An Overview

A number of algorithms for table detection in document images and textual documents emerged from document analysis research. LeNDI's table detection process is based on the bottom-up table detection process described in [KD99, Kie98]. It starts by identifying single words. Then, it groups words in blocks. Finally, it tries to discover the relation between these blocks and see if they are organized in a tabular structure or not. The following describes the algorithms developed and implemented in LeNDI to realize this process.

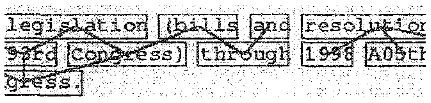
In the bottom-up view of the snapshot, the lowest level is the word, delimited by white space or snapshot boundaries on either side, and constrained to a single row. Figure 4.11 shows the words identified on a portion of the legacy screen snapshot of Figure 4.3.



```
be legislation (bills and res
3 (93rd Congress) through 199
Congress.
```

Figure 4.11. The Identified Words on Part of a Legacy Screen Snapshot.

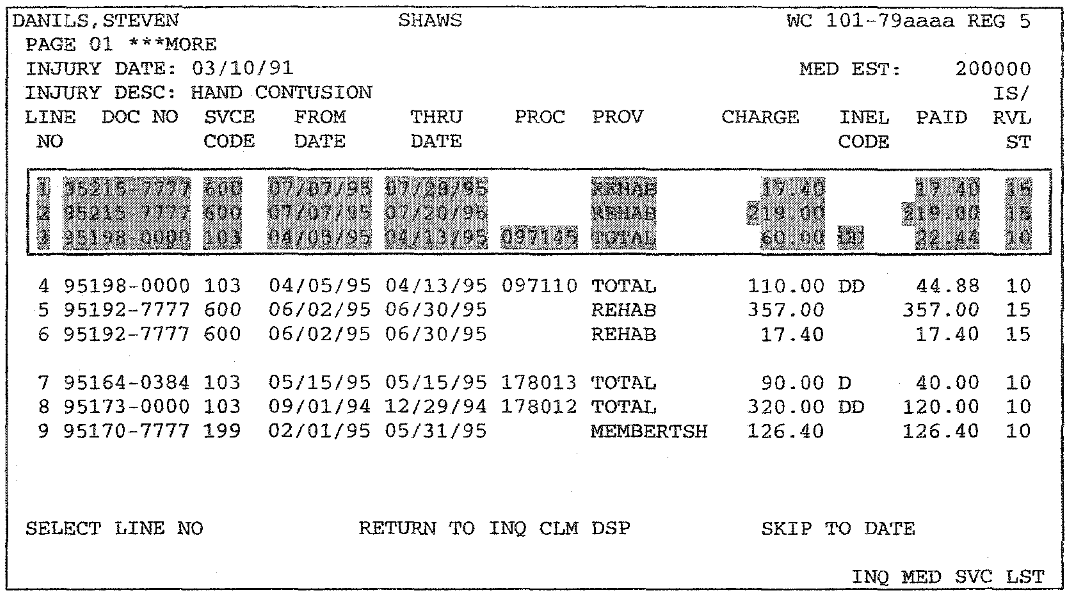
The next level is comprised of blocks, which are constructed from words. Any two words that are vertically adjacent are members of the same block. In order to be vertically adjacent, there must exist at least one column, which both words occupy, and the words must be in consecutive rows. Figure 4.12 shows portions of the two blocks that contain the words shown in Figure 4.11. The relation between all the blocks in the document (snapshot in our case) is then studied to see if they form a table.



```
legislation (bills and resolution
93rd Congress) through 1998 A05th
gress.
```

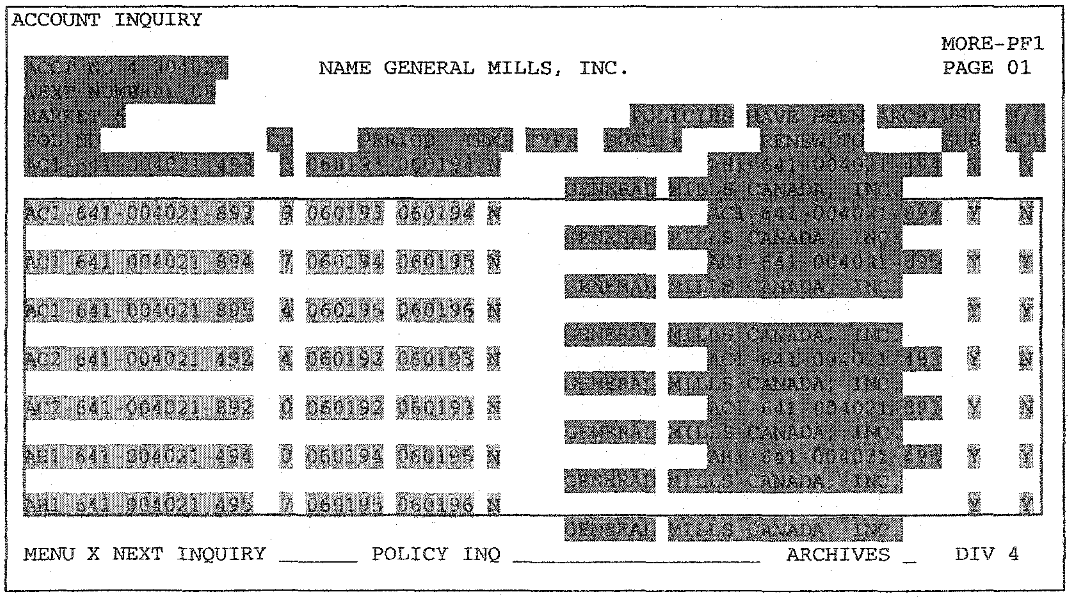
Figure 4.12. Portions of The Identified Blocks on a Legacy Screen Snapshot.

A table is characterized by having several columns of information. Note that a column is generally defined by spanning more than one row, to distinguish it from a text segment that occupies one row. By studying many legacy screen snapshots, one can recognize two major types of tables. There is the single-row record table, whose record of data occupies only one physical row on the snapshot. So, each of its columns contains similar pieces of information, each on a different row (See Figure 4.13). There is also the multiple-row record table, where related information in a record is spread across multiple rows (See Figure 4.14). Thus, a column in this case may contain different types of information. These two types of tables consist of different types of blocks and have unique properties that necessitate separate consideration.



```
DANILS,STEVEN                   SHAWS                         WC 101-79aaaa REG 5
 PAGE 01 ***MORE
 INJURY DATE: 03/10/91                                         MED EST:     200000
 INJURY DESC: HAND CONTUSION                                                   IS/
 LINE  DOC NO   SVCE    FROM      THRU     PROC   PROV       CHARGE   INEL  PAID  RVL
  NO            CODE    DATE      DATE                                CODE         ST

  1 95215-7777 600   07/07/95 07/20/95        REHAB        17.40          17.40  15
  2 95215-7777 600   07/07/95 07/20/95        REHAB       219.00         219.00  15
  3 95198-0000 103   04/05/95 04/13/95 097145 TOTAL        60.00 DD       22.44  10

  4 95198-0000 103   04/05/95 04/13/95 097110 TOTAL       110.00 DD       44.88  10
  5 95192-7777 600   06/02/95 06/30/95        REHAB       357.00         357.00  15
  6 95192-7777 600   06/02/95 06/30/95        REHAB        17.40          17.40  15

  7 95164-0384 103   05/15/95 05/15/95 178013 TOTAL        90.00 D        40.00  10
  8 95173-0000 103   09/01/94 12/29/94 178012 TOTAL       320.00 DD      120.00  10
  9 95170-7777 199   02/01/95 05/31/95        MEMBERTSH   126.40         126.40  10


 SELECT LINE NO            RETURN TO INQ CLM DSP          SKIP TO DATE

                                                               INQ MED SVC LST
```

Figure 4.13. A Single- Row Record Table.



```
ACCOUNT INQUIRY
                                                                     MORE-PF1
 ACCT NO 4 004021             NAME GENERAL MILLS, INC.               PAGE 01
 NEXT NUMERAL 08
 MARKET 4                                    POLICIES HAVE BEEN ARCHIVED  B/I
 POL NR               CD     PERIOD TERM  TYPE  BOND #        RENEWING  SUB  ACT
 AC1-641-004021-493   3 060193 060194 N                 AH1-641-004021-493  Y  N
                                          GENERAL MILLS CANADA, INC.
 AC1-641-004021-893   3 060193 060194 N                 AC1-641-004021-894  Y  N
                                          GENERAL MILLS CANADA, INC.
 AC1-641-004021-894   7 060194 060195 N                 AC1-641-004021-895  Y  Y
                                          GENERAL MILLS CANADA, INC.
 AC1-641-004021-895   4 060195 060196 N                                     Y  Y
                                          GENERAL MILLS CANADA, INC.
 AC2-641-004021-492   4 060192 060193 N                 AC1-641-004021-493  Y  N
                                          GENERAL MILLS CANADA, INC.
 AC2-641-004021-892   0 060192 060193 N                 AC1-641-004021-893  Y  N
                                          GENERAL MILLS CANADA, INC.
 AH1-641-004021-494   0 060194 060195 N                 AH1-641-004021-495  Y  Y
                                          GENERAL MILLS CANADA, INC.
 AH1-641-004021-495   7 060195 060196 N                                     Y  Y
                                          GENERAL MILLS CANADA, INC.
 MENU X NEXT INQUIRY ______ POLICY INQ ____________________ ARCHIVES _   DIV 4
```

Figure 4.14. A Multiple-Row Record Table.

Many single-row record tables have one word per row. To find such tables, LeNDI needs to look for blocks of text where each word is connected to at most one word in each vertical direction. These blocks are dubbed "thin blocks" to indicate their special nature, which is that they contain one word per row. If these "thin" blocks are found in reasonable amounts across consecutive rows on the screen, the area should be declared a table. The area would be limited horizontally by the leftmost and rightmost thin blocks and vertically by the span of the rows throughout which thin blocks can be found. For example, the boxed area in Figure 4.13 could be labeled as a table, because there are 9 thin blocks that span at least 3 shared rows. Between the leftmost and rightmost thin blocks, this area may also contain non-thin blocks. Note that the detected table does not extend past the first and last thin blocks, even if the actual table does. The next subsection introduces the necessary default or user-defined parameters for defining the criterion of accepting or rejecting a group of thin blocks as a table.

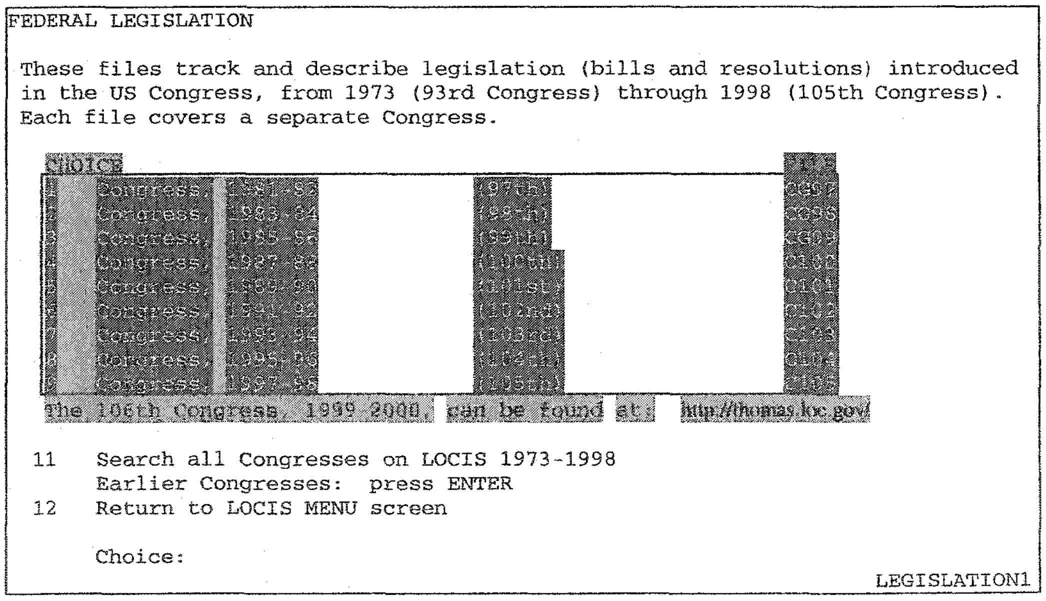
To detect multiple-row record tables, LeNDI compares the starting column of the top row (top-left) of the blocks that start on the same row with those that start on other rows. These blocks do not have to have the same dimensions; however, the two rows should have enough number of blocks that begin there. There should also exist a certain number of blocks that have identical borders on the left side. If this condition is met then the area that contains the table (the smallest rectangle that contains all the blocks) is identified as such. For example, the boxed area in Figure 4.14 would be labeled as a table, despite the large block in the middle that would confuse other table identification attempts. Note that in Figure 4.14, the blocks with light gray background are those that were detected as part of a table. They meet the conditions described above. The ones with dark gray background do not contribute to table detection, as they do not fulfill the necessary conditions. Only the parts of these blocks that fall inside the table borders are considered part of the table. The first record of the actual table on the snapshot is not detected as part of the table.

One of the problems identified in [KD99, Kie98] is the effect of common headers (or footers) on blocks. This effect is shown in Figure 4.15. Essentially, it causes multiple thin blocks to be tied together by a header or footer that spans multiple columns. This can cause problems to LeNDI's table detection strategy. One solution to this problem is to consider instances of thin sub-blocks within larger blocks and use these as if they were separate. To identify the sub-blocks, one considers groups of words inside a block that are vertically connected with one another and disconnected from the surrounding text. If they span sufficient rows, then they are considered thin sub-blocks. Then, one can apply Algorithm 4.6b for single-row record table detection to both the original thin blocks and the new thin sub-blocks, to see if there is a table.

4.4.2.2 Table Detection: Process and Algorithms

The sequel gives an overview of the application of the above methods in LeNDI, followed by the algorithms used. LeNDI needs to prioritize the above ideas before applying them. First, LeNDI divides a given presentation space into words and then groups these words into blocks (Algorithm 4.6a). Second, as tables composed of thin blocks are the most prevalent and the least computationally intensive, LeNDI tries to identify this type first (Algorithm 4.6b). Since LeNDI is designed to deal with a wide range of legacy UI styles, it offers the user some control over this process. Specifically, the user is allowed to override the default values of LeNDI for these two parameters:

1. The minimum number of blocks (columns) that must exist in a table. The default is 3.
2. The minimum number of rows that these blocks must all span. The default is 3.



```
FEDERAL LEGISLATION

 These files track and describe legislation (bills and resolutions) introduced
 in the US Congress, from 1973 (93rd Congress) through 1998 (105th Congress).
 Each file covers a separate Congress.

   CHOICE                                                    FILE
   1   Congress, 1981-82          (97th)                     C097
   2   Congress, 1983-84          (98th)                     C098
   3   Congress, 1985-86          (99th)                     C099
   4   Congress, 1987-88          (100th)                    C100
   5   Congress, 1989-90          (101st)                    C101
   6   Congress, 1991-92          (102nd)                    C102
   7   Congress, 1993-94          (103rd)                    C103
   8   Congress, 1995-96          (104th)                    C104
   9   Congress, 1997-98          (105th)                    C105
   The 106th Congress, 1999-2000, can be found at:   http://thomas.loc.gov/

  11    Search all Congresses on LOCIS 1973-1998
        Earlier Congresses:  press ENTER
  12    Return to LOCIS MENU screen

        Choice:
                                                              LEGISLATION1
```

Figure 4.15. Embedded Thin Blocks

Third, if the second step fails to find a table, LeNDI gathers thin sub-blocks from the set of blocks discovered (Algorithm 4.6c). Then, it reapplies Algorithm 4.6b used in the second step above. Beside the above parameters, the user can control a parameter that defines when to consider a thin sub-block separate from its parent block:

3. The minimum number of rows a thin sub-block must span to be considered separate from its parent. The default is 3.

Fourth, if the third step fails too, LeNDI applies the top-left matching algorithm (Algorithm 4.6d) to the collection of blocks. This algorithm would reuse the parameters: the minimum number of blocks (columns) and the minimum number of rows. The first defines the minimum number of matching blocks a table must have between two rows. The second is used to indicate the minimum number of rows that must have matching blocks. In the following, Algorithms 4.a to 4.d, which are used by LeNDI to implement this process, are presented.

Algorithm 4.6a starts by creating empty lists to store the words and the blocks identified on the given presentation space (steps 1 and 2). Note that blocks are graphs whose nodes are words. Suitable classes or data structures need to be created. But these details are not shown in the abstract algorithms given here. Step 3 extracts the individual words from the presentation space and stores them in *Word List*. What are actually stored are the words' dimensions, i.e., row, starting column and length. For every word extracted, step 4 tries to find all the blocks that it belongs to, i.e., blocks that the word is adjacent to at least one word in each of them (step 4.b) and merges them together (4.c). But if the word does not belong to any block, then it is put in a new block, which is added to the *Block List* (step 4.e).

Algorithm 4.6b aims to find the first table it encounters that meets the default or user-set criterion for tables. This algorithm does not aim to find all the tabular structures on the snapshot or discover their relation to one another, e.g., if they are parts of the same table but are fragmented from each other.

Step 1 in Algorithm 4.6b creates a list to store thin blocks, *Thin Blocks*. Step 2 marks the non-thin blocks. It identifies them as the ones having at least one word that is adjacent to at least two words from above or two words from below. Then step 2.c adds blocks that passed the thinness test without getting marked to *Thin Blocks*. Steps 3 and 4 retrieve the parameters that define how many thin blocks (*min # of columns)* are needed and how many mutual rows (*min # of rows*) they need to span in order to be considered a table.

Step 5 goes over every set of rows of height *min # of rows* to see how many thin blocks span this set of rows, i.e., start above or at the first row and end below or at the last row. Step 5.c performs this check, counts such thin blocks and calculates the dimensions of the minimum rectangle that covers these thin blocks, which is considered as the dimension of the table formed of these blocks if any. Step 5.d checks if enough thin blocks (greater than or equal *min # of columns*) span the given set of rows. If yes, then it reports the current table dimensions and terminates the algorithm. If the current set of rows does not have enough thin blocks to form a table, step 5 moves one row down, takes the next *min # of rows* rows and repeats the check of which thin blocks span this set of rows. If the test fails for all sets of *min # of rows* consecutive rows, then step 6 reports that no table was found using thin-block analysis.

Algorithm 4.6a: Breaking a Presentation Space into Blocks of Words
Input: A presentation space, *Pres Space.*
Output: A list of all word blocks in *Pres Space.*
Steps:

```
1. Create a new list, Word List
2. Create a new list, Block List

3. For every row in the Pres Space
     • Break row into words
     • Store the words' coordinates and lengths in Word List

4. For every word in Word List
   a. Success = FALSE
   b. For every block in Block List
        • If word is adjacent to block then
          ❑ Add word to block
          ❑ Mark block
          ❑ Success = TRUE

   c. Merge all marked blocks together
   d. Unmark all marked blocks
   e. If Success == FALSE then
        • Create a new block, New Block
        • Add word to New Block
        • Add New Block to Block List
```

Algorithm 4.6a. Breaking a Presentation Space into Blocks of Words

```
Algorithm 4.6b: Table Detection Using Thin Blocks
Input: A list of blocks, Block List and the presentation space's number of rows, length.
Output: The dimensions of the table formed of the input blocks if any.
Steps:
1. Create a new list, Thin Blocks
2. For all the blocks in Block List
   a. Get current block
   b. For every word in current block
      • If connected with two words from above or two words from below then
        ❑ mark current block as non-thin block
   c. If current block is thin then add it to Thin Blocks
3. Retrieve the parameter min # of rows from database
4. Retrieve the parameter min # of columns from database
5. For i = 1 to length - min # of rows +1
   a. column count = 0
   b. Create new Dimensions table dimensions
   c. For every block in Thin Blocks
      • If (first row in block ≤ i) && (last row in block ≥ i + min # of rows -1) then
        ❑ column count ++
        ❑ Update table dimensions to include block
   d. If (column count ≥ min # of columns) then
      • Report table dimensions
      • Return message "Table found"
6. Return message "Table NOT found"
```

Algorithm 4.6b. Table Detection Using Thin Blocks

Algorithm 4.6c discovers thin sub-blocks in a given block, *block*. Step 1 creates a list to store the discovered thin sub-blocks, *Thin Sub-block List*. Step 2 retrieves the parameter that defines how many rows thin sub-blocks should span in order to be considered and analyzed independent from its parent block. Step 3 marks all the words with only one adjacent word from the row above and one from the row below, which are candidates for being in one of the thin sub-blocks. Step 4 loops while there are still marked words in *block*. Steps 4.a to 4.c get the next marked word and create a new sub-block for it, *sub-block*. Steps 4.d and 4.e get the upper and lower adjacent words of the current marked word. Step 4.f grows *sub-block* from above by looping as long as there are more upper adjacent marked words, fetching these words and adding them to *sub-block*. Step 4.g deals with the case when the top word of a sub-block has to adjacent words from above, and hence, is not marked. Step 4.g still adds such word to *sub-block*. Figure 4.16 shows two cases, one when step 4.g would not apply (left) and another when it applies (right). In the left case, all the words of both thin sub-blocks are marked as they all have at most one adjacent word from each direction. In the right case, the top word of the right block is unmarked as it has two adjacent words from above. Step 4.g would still add this word to the thin sub-block. Steps 4.h and 4.i are similar to 4.f and 4.g but grow *sub-block* from below. Step 4.j adds *sub-block* to *Thin Sub-block List* if it spans at least the minimum number of rows parameter. Step 4.k unmarks all the words in *sub-block*. Finally, step 5 reports the discovered thin sub-blocks.

Algorithm 4.6c: Discovering Thin Sub-blocks inside a Block
Input: A block, *block*.
Output: A list of all the thin sub-blocks in *block*.
Steps:

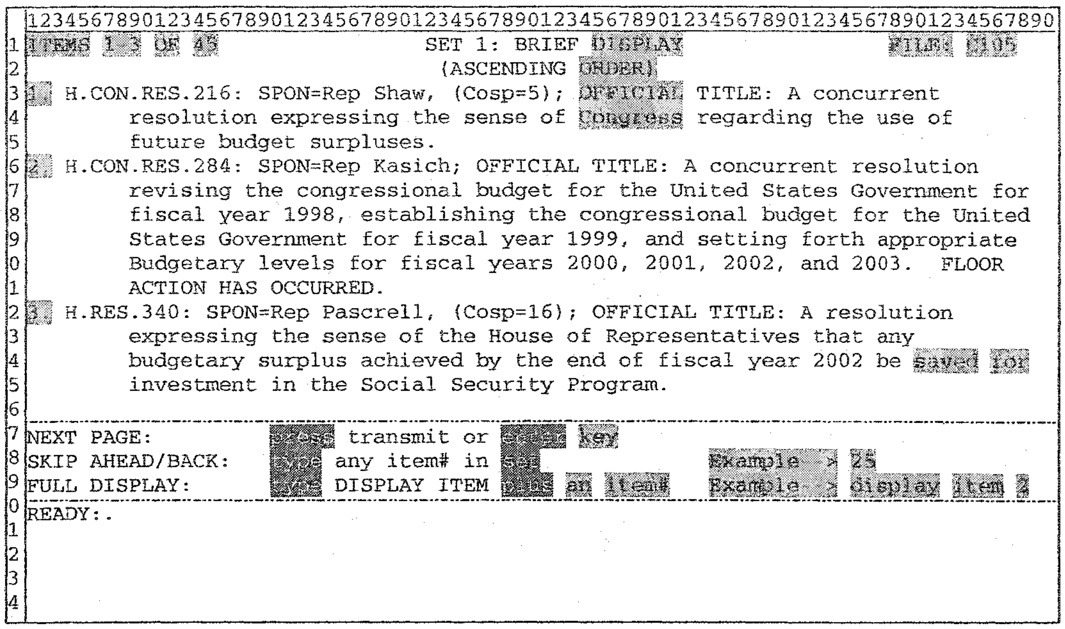
```
1. Create a new list, Thin Sub-block List
2. Retrieve the parameter min # of rows of a thin sub-block
3. For every word in block
     • If word is adjacent with at most one word from below and one word from above
         □ then mark word
4. While there are still some marked words in block do
   a. Get next marked word
   b. Create new Block sub-block
   c. Add marked word to sub-block
   d. Get upper adjacent word to marked word
   e. Get lower adjacent word to marked word
   f. While upper adjacent word is marked
        • Add upper adjacent word to sub-block
        • Get next adjacent word to upper adjacent word
        • upper adjacent word = next adjacent word
   g. If upper adjacent word has one adjacent word from below then
        • Add upper adjacent word to sub-block
   h. While lower adjacent word is marked
        • Add lower adjacent word to sub-block
        • Get next adjacent word to lower adjacent word
        • lower adjacent word = next adjacent word
   i. If lower adjacent word has one adjacent word from above then
        • Add lower adjacent word to sub-block
   j. If (height (sub-block) ≥ min # of rows of a thin sub-block) then add sub-block to
      Thin Sub-block List
   k. Unmark all words in sub-block
5. Report Thin Sub-block List
```

Algorithm 4.6c. Discovering Thin Sub-blocks inside a Block

CHOICE
CHOICE Of CONGRESS
Congress,

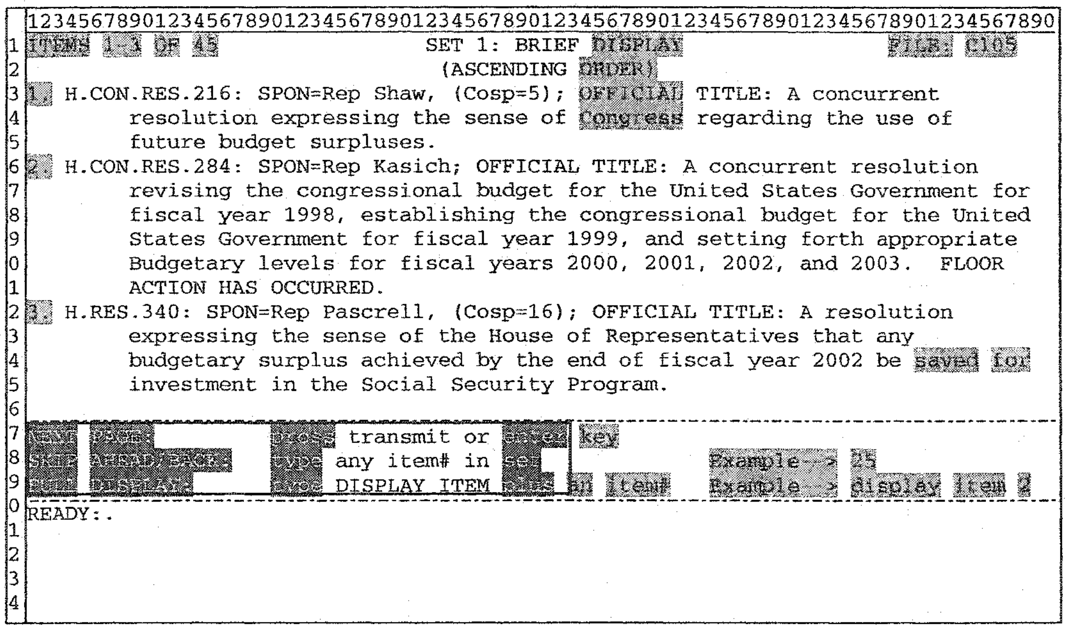
Figure 4.16. Two Cases where Step 4.g in Algorithm 4.6c Is Skipped (Left) and Applied (Right). Gray Words Have 1 or No Adjacent Words from below and above.

To demonstrate the application of Algorithms 4.7b and 4.7c, an example is given in Figure 4.17. In Figure 4.17a, Algorithm 4.6b was applied to discover thin blocks, which are shown in gray. Then the algorithm proceeded to detect the relation between these blocks. The default value of 3 was used for all the parameters, which are the minimum number of rows for a table, the minimum number of thin blocks (columns) and the minimum number of rows for a thin sub-block to be considered independent form its parent. The sliding window with dashed frame and height 3 kept sliding down from the top of the snapshot with no success in detecting a table. Figure 4.16a shows when the algorithm was analyzing rows 17 to 19. The thin blocks with dark background meet the condition of vertically spanning at least the three rows under analysis, but there are only two of them. So they do not form a table. The blocks with light gray background do not meet the condition. Figure 4.17b shows the same analysis after applying Algorithm 4.6c to discover thin sub-blocks and include them in the reapplication of Algorithm 4.6b. This time, there are four thin blocks and sub-blocks that vertically span the lines currently under study in the dashed frame. The conclusion is that the area in the solid line frame, which is the minimum rectangle that covers the thin blocks of the table, contains a table, according to the criterion defined by the parameters used.



```
1234567890123456789012345678901234567890123456789012345678901234567890123456789 0
ITEMS 1-3 OF 45                    SET 1: BRIEF DISPLAY                FILE: C105
                                    (ASCENDING ORDER)
1. H.CON.RES.216: SPON=Rep Shaw, (Cosp=5); OFFICIAL TITLE: A concurrent
        resolution expressing the sense of Congress regarding the use of
        future budget surpluses.
2. H.CON.RES.284: SPON=Rep Kasich; OFFICIAL TITLE: A concurrent resolution
        revising the congressional budget for the United States Government for
        fiscal year 1998, establishing the congressional budget for the United
        States Government for fiscal year 1999, and setting forth appropriate
        Budgetary levels for fiscal years 2000, 2001, 2002, and 2003.  FLOOR
        ACTION HAS OCCURRED.
3. H.RES.340: SPON=Rep Pascrell, (Cosp=16); OFFICIAL TITLE: A resolution
        expressing the sense of the House of Representatives that any
        budgetary surplus achieved by the end of fiscal year 2002 be saved for
        investment in the Social Security Program.

NEXT PAGE:             press transmit or enter key
SKIP AHEAD/BACK:       type any item# in set          Example > 25
FULL DISPLAY:          type DISPLAY ITEM plus an item#  Example > display item 2
READY:.
```

(a) The application of thin block analysis (Algorithm 4.6b) does not detect any table.



```
1234567890123456789012345678901234567890123456789012345678901234567890123456789 0
ITEMS 1-3 OF 45                    SET 1: BRIEF DISPLAY                FILE: C105
                                    (ASCENDING ORDER)
1. H.CON.RES.216: SPON=Rep Shaw, (Cosp=5); OFFICIAL TITLE: A concurrent
        resolution expressing the sense of Congress regarding the use of
        future budget surpluses.
2. H.CON.RES.284: SPON=Rep Kasich; OFFICIAL TITLE: A concurrent resolution
        revising the congressional budget for the United States Government for
        fiscal year 1998, establishing the congressional budget for the United
        States Government for fiscal year 1999, and setting forth appropriate
        Budgetary levels for fiscal years 2000, 2001, 2002, and 2003.  FLOOR
        ACTION HAS OCCURRED.
3. H.RES.340: SPON=Rep Pascrell, (Cosp=16); OFFICIAL TITLE: A resolution
        expressing the sense of the House of Representatives that any
        budgetary surplus achieved by the end of fiscal year 2002 be saved for
        investment in the Social Security Program.

NEXT PAGE:             press transmit or enter key
SKIP AHEAD/BACK:       type any item# in set          Example > 25
FULL DISPLAY:          type DISPLAY ITEM plus an item#  Example > display item 2
READY:.
```

(b) Reapplying thin block and sub-block analysis (Algorithm 4.6b), after identifying thin sub-blocks using Algorithm (4.6c), detects the table in solid border.

Figure 4.17. An Example Application of Algorithms 4.6b and 4.6c

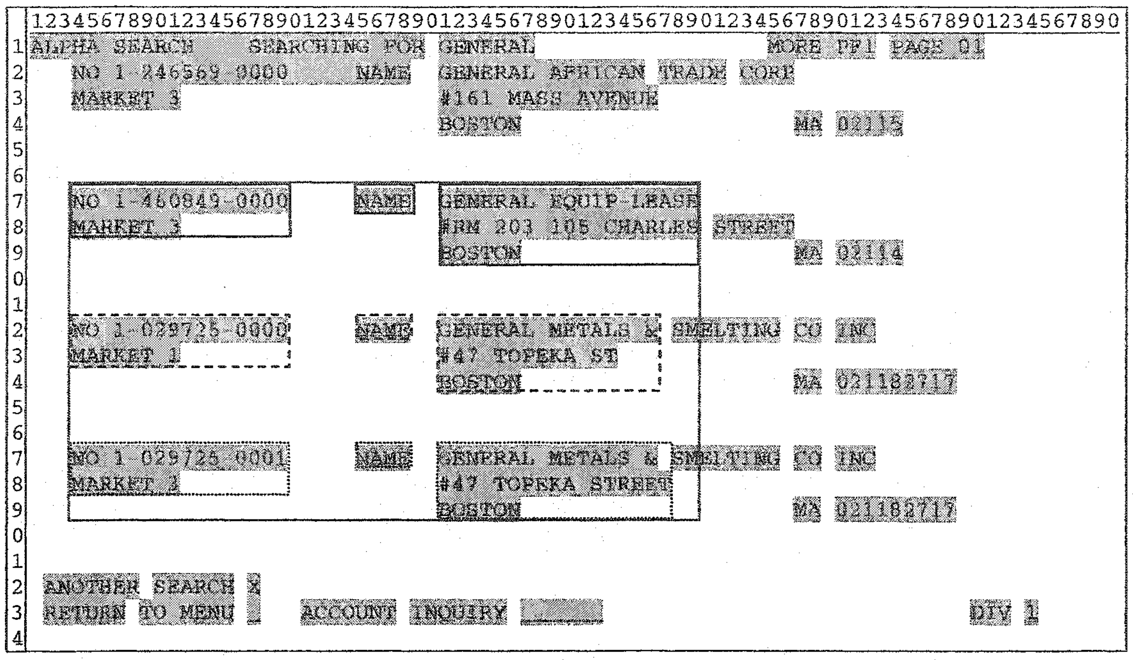
If table discovery by analysis of thin-blocks and sub-blocks fails, LeNDI tries to detect any existing multiple-row record table using its block top-left matching algorithm (Algorithm 4.6d). The idea of the algorithm is to iterate over the rows of the presentation space. For every current row, it tries to discover if there is a table that starts there. This is done by storing the blocks that start on this row. Then, for every subsequent row, the algorithm collects the blocks that start on this next row and checks if they qualify to be part of a potential table that starts on the current row. This is done be comparing if enough blocks on both rows have the same left column border. If enough subsequent rows qualify, then the algorithm declares that a table is found, reports the table dimensions and terminates.

Algorithm 4.6d takes as input a list of blocks resulting from Algorithm 4.6a and the number of rows in the presentation space analyzed. It outputs the dimensions of the first table formed of the input blocks that meets the user criterion, defined by the minimum number of rows and columns parameters. Step 1 creates a new object, *Table Dimensions*, to store the dimensions of any table discovered. Steps 2 and 3 retrieve the parameters *min # of rows* and *min # of columns*. Step 4 iterates over every row *i* and stops when a table is found. Steps 3.2 and 3.3 store, in a new block list *Table BL*, all blocks that start on the current row. If the number of blocks starting on the current row is less than *min # of columns*, step 3.4 ends the current iteration and moves to the next row. Step 3.5 iterates over every row *j* subsequent to the current row and checks if it qualifies for being part of a table that starts at *i*. Steps 3.5.1 and 3.5.2 store all the blocks that start on *j* in a new block list *Candidate BL*. Step 3.5.3 checks if *j* has enough columns (blocks) compared to *min # of columns*. If yes, then it checks how many of these columns has left boundaries that match some block of *Table BL*. If the number of matching blocks is ≥ *min # of columns*, then this row is considered part of a potential table and *Num Qualifying Rows* is incremented and *Table Dimensions* is updated to include all the matching blocks. After iterating over all the subsequent rows of *i*, step 3.6 checks if the number of qualified rows is ≥ *min # of rows*. If yes, it reports *Table Dimension*s, returns a success message and terminates the algorithm. If no, step 3.7 resets *Table Dimensions*, and another iteration starts to try to discover a table starting at row *i*+1. If no table was discovered that starts at any row, step 5 terminates the algorithm with a failure message.

```
Algorithm 4.6d: Block Top-left Matching Algorithm
Input: List of blocks, Block List, and the number of presentation space rows, length.
Output: The dimensions of the table formed of (some of) the input blocks, if any.
Steps:
1. Create new Dimension Table Dimensions
2. Retrieve the parameters min # of rows and min # of columns from database
3. For i = 1 to length
   3.1. Num Qualifying Rows = 1
   3.2. Create a new block list, Table BL
   3.3. For every block in Block List
        • If (top row of block = i) then
          ❑ Add block to Table BL
   3.4. If (sizeof (Table BL) < min # of columns) then Continue
   3.5. For j = i+1 to length
        3.5.1 Create a new block list, Candidate BL
        3.5.2 For every block in Block List
              ❑ If (top row of block = j) then
                ▪ Add block to Candidate BL
        3.5.3 If (sizeof (Candidate BL) < min # of columns) then
              ❑ matching Columns = the # of blocks in Candidate BL whose leftmost
                                   column matches that of a block in Table BL
              ❑ If (matching Columns ≥ min # of columns) then
                ▪ Num Qualifying Rows ++
                ▪ Update Table Dimensions to include all blocks in Table BL and
                  Candidate BL whose leftmost columns match
   3.6. If (Num Qualifying Rows ≥ min # of rows) then
        • Report Table Dimensions
        • Return message "Table found"
   3.7. Else Reset Table Dimensions
4. Return message "Table NOT found"
```

Algorithm 4.6d. Block Top-left Matching Algorithm

Figure 4.18 illustrates the application of Algorithm 4.6d. The minimum number of rows and columns sought was three. The algorithm failed to discover a table until row 7. Three blocks start at row 7. Their dimensions (top row, left column, bottom row, right column) are shown in the table of Figure 6.18(b). Then, the algorithm analyzed the consecutive rows and discovered that each of rows 12 and 17 has three blocks with the same left boundary as the three blocks of row 7 (shown in bold font). The algorithm concluded that rows 7, 12 and 17 form a table whose dimensions are (7,4,19,49), i.e., it includes all the blocks with matching left columns.



```
 12345678901234567890123456789012345678901234567890123456789012345678901234567890
1ALPHA SEARCH     SEARCHING FOR GENERAL                         MORE PF1 PAGE 01
2   NO 1-246569-0000      NAME  GENERAL AFRICAN TRADE CORP
3   MARKET 3                    #161 MASS AVENUE
4                               BOSTON                            MA 02115
5
6
7   NO 1-460849-0000      NAME  GENERAL EQUIP-LEASE
8   MARKET 3                    #RM 203 105 CHARLES STREET
9                               BOSTON                            MA 02114
0
1
2   NO 1-029725-0000      NAME  GENERAL METALS & SMELTING CO INC
3   MARKET 1                    #47 TOPEKA ST
4                               BOSTON                            MA 021182717
5
6
7   NO 1-029725-0001      NAME  GENERAL METALS & SMELTING CO INC
8   MARKET 3                    #47 TOPEKA STREET
9                               BOSTON                            MA 021182717
0
1
2 ANOTHER SEARCH X
3 RETURN TO MENU _    ACCOUNT INQUIRY ______                          DIV 1
4
```

(a) An example snapshot with a multiple-row record table detected.

Row #	The Dimensions Of The Blocks Starting On This Row					
7	(7,**4**,8,19)	(7,**25**,7,28)	(7,**31**,9,49)			
8				(8,51,8,56)		
9					(9,67,9,68)	(9,70,9,74)
10						
11						
12	(12,**4**,13,19)	(12,**25**,12,28)	(12,**31**,14,46)	(12,48,12,55)	(12,67,12,68)	(12,70,12,72)
13						
14					(14,67,14,68)	(14,70,14,78)
15						
16						
17	(17,**4**,18,19)	(17,**25**,17,28)	(17,**31**,19,47)	(17,48,17,55)	(17,67,17,68)	(17,70,17,72)

(b) The dimensions of the blocks starting or rows 7 to 17.

Figure 4.18. An Example Application of Algorithm 4.6d

4.4.2.3 List Detection

If LeNDI fails to recognize any tabular structure on a snapshot, it tries to discover if the snapshot contains a list. A list is characterized by the following:

1. There exists a column of numbers in the left half of the snapshot. This can be decided from the "numbers binary vertical profile".
2. The column contains "enough" numbers, which are neither real numbers nor dates or times. The user need to define how many numbers are enough by setting up the parameter "minimum list length", or LeNDI will use the default value of 3.

LeNDI employs the heuristic Algorithm 4.7 below to discover the existence of a list, if any, on a given snapshot. Steps 1 and 2 build the "numbers binary vertical profile" of the given snapshot presentation space and store it in a variable, *Nums Profile.* Step 3 terminates the algorithm if no numerical content is found in the left half of *Nums Profile.* Otherwise, step 4 gets the location of the left most sequence of "1"s in *Nums Profile*, which is assumed to be corresponding to the list indices, if any. Steps 5 to 8 extract the snapshot columns corresponding to this sequence of ones, cut the top and bottom of these columns using the setup parameters upper and lower vertical cuts and finally store them in *Strip*. Step 9 extracts all the numbers in *Strip* and stores them in *Numbers List.* It excludes numbers that are part of a date or a time or part of a real number. Steps 10 and 11 terminate the process if the length of *Numbers List* is less than the setup parameter: minimum list length. Reaching step 12 means that a list was found. Step 12 collects its attributes, the list order, first element, size, increment, first element's row and left boundary and right boundary of *Strip*. Steps 13 and 14 report these attributes and return a message that a list was found.

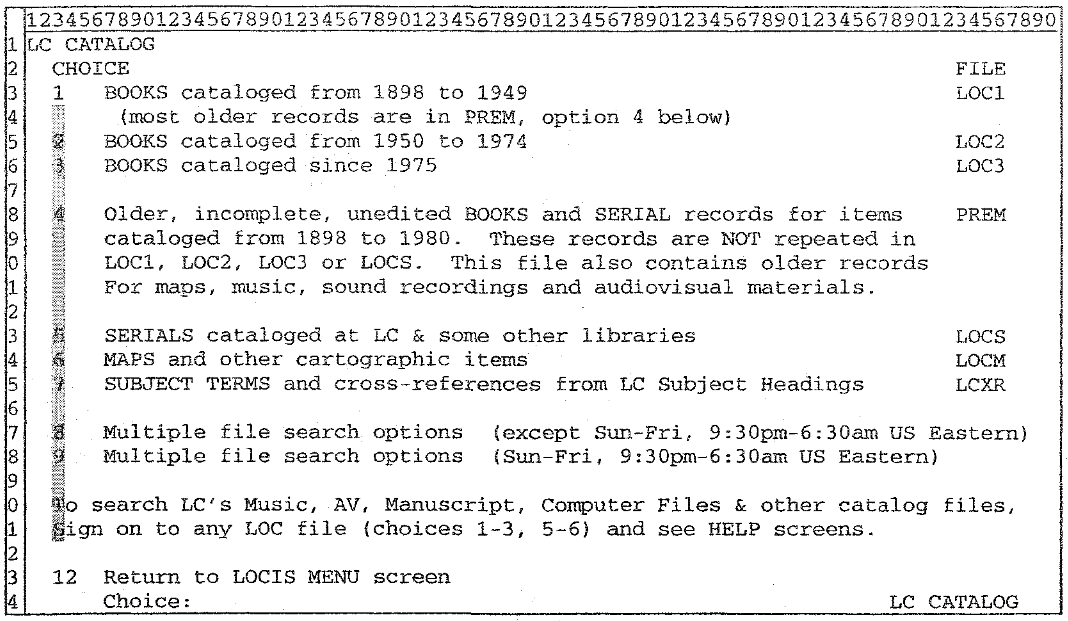
To better understand Algorithm 4.7, an example is given in Figure 4.19. The upper of Figure 4.19 shows a legacy screen snapshot. The bottom shows a projection of the numerical content of the snapshot on the horizontal axis after cutting the top "Upper Vertical Cut" lines and the bottom "Lower Vertical Cut" liners off the snapshot and the Numbers Vertical Profile of the snapshot in binary format. The gray strip on the upper figure is the strip of interest that Algorithm 4.7 extracted and analyzed for this snapshot after examining the profile at the bottom. The list indices retrieved from this strip are 2, 3, 4, 5, 6, 7, 8 and 9.

Algorithm 4.7: List Detection

Input: The presentation space, *Pres Space*, of a given snapshot.
Output: A message indicating if a list was detected or not and list specifications, if any.
Steps:

```
1  Call the Numbers Binary Vertical Profile Algorithm (4.5), with Pres Space as input.
2  Store the profile in binary format in Nums Profile
3  If the first half of Nums Profile is all zeros then Return message "No List"
4  Else get the start location, start loc, and length, len, of the leftmost non-zero sequence
   in Nums Profile
5  Extract from Pres Space the column start loc and len-1 consecutive columns and
   store them in Strip
6  Retrieve the Upper Vertical Cut and Lower Vertical Cut from the database
7  Cut the Upper Vertical Cut lines from the top of Strip
8  Cut the Lower Vertical Cut lines from the bottom of Strip
9  For every line in Strip
   a. Extract the first sequence of digits and store it in number
   b. If number is empty then discard number
   c. Else If number is part of a date, time or real number, i.e., if it is
         • Succeeded or preceded by a slash "/"
         • Succeeded or preceded by a slash ":"
         • Succeeded by a dot and a digit, e.g., ".6" or preceded by a dot "."
      then discard number
   d. Else store number in Numbers List
10 Retrieve the setup parameter Min List Length
11 If length(Number List) < Min List Length then Return message "No List"
12 Else
   a. Ascending = Descending = Equal = Increment = 0
   b. For i =1 to length (Number List) - 1
      • If Numbers List [i] < Numbers List [i+1] then Ascending++
      • If Numbers List [i] > Numbers List [i+1] then Descending++
      • Else Equal++
   c. List Order = The 1st letter of the biggest of Ascending, Descending and Equal, i.e.
      A, D or E.
   d. First Element  = Number List [1]
   e. Size           = length (Number List)
   f. Increment      = (Number List [Size]- Number List [1]) / (Size – 1)
   g. 1st Element's Row    = The row on which the first number in the list.
   h. Left Boundary        = start loc
   i. Right Boundary       = start loc + len - 1
13 Report List Order, First Element, Size, Increment, 1st Element's Row, Left Boundary
   and Right Boundary
14 Return message "A List Was Detected"
```

Algorithm 4.7. List Detection Algorithm



```
LC CATALOG
  CHOICE                                                            FILE
  1   BOOKS cataloged from 1898 to 1949                             LOC1
        (most older records are in PREM, option 4 below)
  2   BOOKS cataloged from 1950 to 1974                             LOC2
  3   BOOKS cataloged since 1975                                    LOC3

  4   Older, incomplete, unedited BOOKS and SERIAL records for items   PREM
      cataloged from 1898 to 1980.  These records are NOT repeated in
      LOC1, LOC2, LOC3 or LOCS.  This file also contains older records
      For maps, music, sound recordings and audiovisual materials.

  5   SERIALS cataloged at LC & some other libraries                LOCS
  6   MAPS and other cartographic items                             LOCM
  7   SUBJECT TERMS and cross-references from LC Subject Headings   LCXR

  8   Multiple file search options  (except Sun-Fri, 9:30pm-6:30am US Eastern)
  9   Multiple file search options  (Sun-Fri, 9:30pm-6:30am US Eastern)

  To search LC's Music, AV, Manuscript, Computer Files & other catalog files,
  Sign on to any LOC file (choices 1-3, 5-6) and see HELP screens.

  12  Return to LOCIS MENU screen
      Choice:                                                  LC CATALOG
```

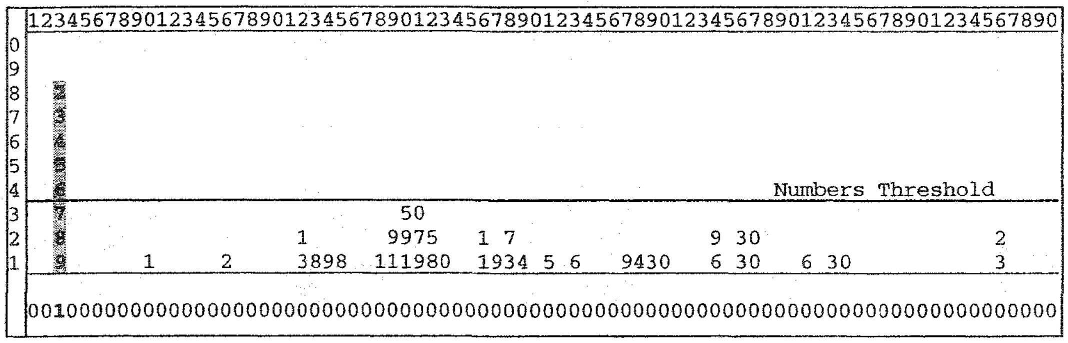


Figure 4.19. An Example Legacy Screen Snapshot (10), Its Vertical Projection and Profile of Its Numerical Content and The Detected List Information Strip (Gray).

4.4.2.4 Feature 7-1 and Feature 7-2: Layout Classification and Specifications

Two features are derived from the layout classification analysis described above. The first is Feature 7-1, which is a single character that describes the layout structure, if any. It takes the vale 'T' if a table was discovered, 'L' if a list was discovered and "" (Blank) if neither a table nor a list was discovered. Feature 7-2 is a multi-part description for the structure discovered if any. For a table, it describes the following:

- Table Start Column
- Table Start Row
- Table Width
- Table Height

- Number of blocks used to detect the table. Note that there may be extra blocks inside the table area that did not contribute to table detection as in Figure 4.25.

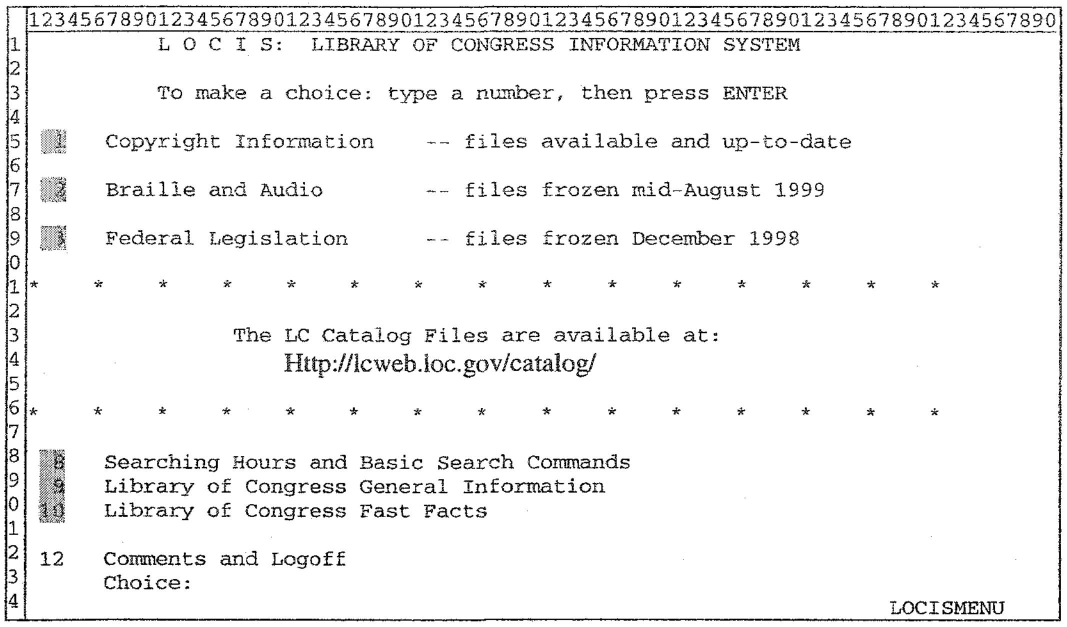
For a list, it describes the following:

- List Order
- First Element
- Number of Elements
- Increment
- First Element's Row
- Left Boundary
- Right Boundary

LeNDI uses binary comparison for Feature 7-1. For Feature 7-2, LeNDI does discrete comparison by comparing the two values part by part. Then, it reports the ratio of the number of matching parts to the total number of parts (5 for a table and 7 for a list).

4.4.2.5 Table and List Detection Examples

This subsection provides some examples to show what layout description can be discovered by Algorithms 4.6 and 4.7 and what the extracted Features 7-1 and 7-2 are for each case. These examples use the sample snapshots used as examples in subsection 4.2.3. The examples are shown in Figures 4.20 to 4.25. The default value of 3 is used for the three setup parameters of table detection and for the only setup parameter used in list detection. On each snapshot, the layout structure discovered by LeNDI, if any, is marked with light and/or dark gray. A description of the discovered structure and the corresponding values of Features 7-1 and 7-2 are given. Note that LeNDI starts the indices of the presentation space columns and rows with zero, while they start with one in the given example for ease of understanding. So, when the "table start column" attribute of a table is 3 in the examples below as discovered by LeNDI, this means on the corresponding presentation space shown the table starts at column 4. Hence, the last part of Feature 7-2 records "4" as the number of columns or blocks of the table. In Figure 4.20, the 3 bottom rows of the snapshot are cut while building the "numbers vertical profile" for list detection. Hence, the menu choice "12" corresponding to "Comments and Logoff" was not discovered as an item on the list. In Figure 4.25, the table detected was detected based on the discovery and the relation between the four thin blocks shown. The fifth non-thin block that is part of the table did not contribute to the detection process. One can argue against that the structure detected on Figure 4.25 as a table. But since the user criterion required only 3 columns and 3 rows to recognize a table, it was detected.

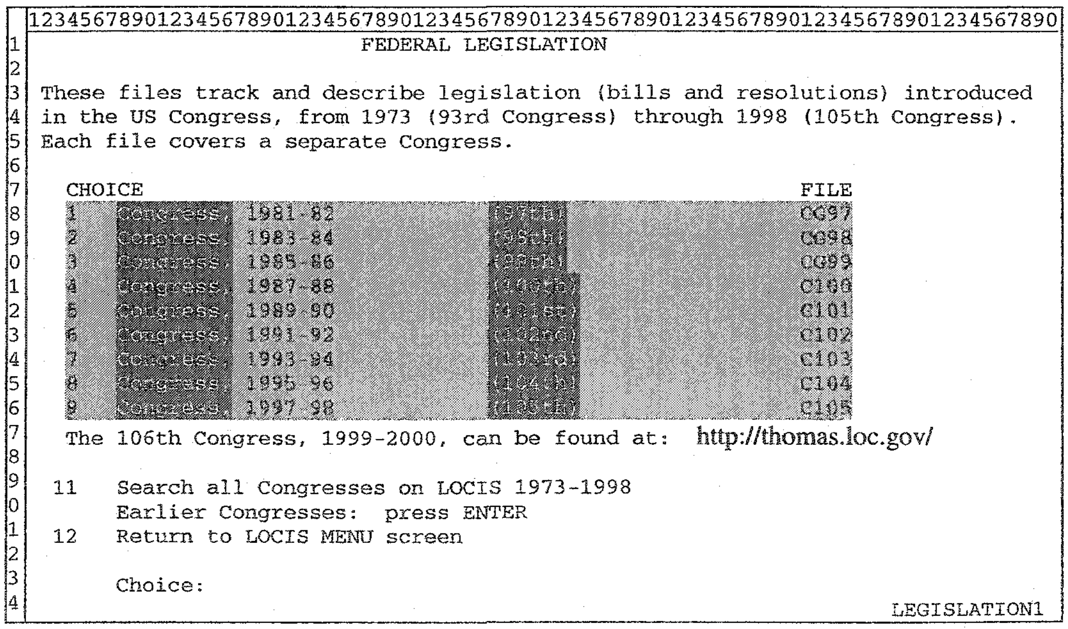


```
 1234567890123456789012345678901234567890123456789012345678901234567890123456789
1          L O C I S:  LIBRARY OF CONGRESS INFORMATION SYSTEM
2
3          To make a choice: type a number, then press ENTER
4
5  1   Copyright Information    -- files available and up-to-date
6
7  2   Braille and Audio        -- files frozen mid-August 1999
8
9  3   Federal Legislation      -- files frozen December 1998
0
1*    *    *    *    *    *    *    *    *    *    *    *    *    *    *
2
3                The LC Catalog Files are available at:
4                    Http://lcweb.loc.gov/catalog/
5
6*    *    *    *    *    *    *    *    *    *    *    *    *    *    *
7
8  8   Searching Hours and Basic Search Commands
9  9   Library of Congress General Information
0 10   Library of Congress Fast Facts
1
2 12   Comments and Logoff
3      Choice:
4                                                                  LOCISMENU
```

List Order	A
First Element	1
Number of Elements	6
Increment	1
1st Element's Row	4
Left Boundary	1
Right Boundary	2

Feature 7-1	L
Feature 7-2	A_1_6_1_4_1_2_

Figure 4.20. An Example Legacy Screen Snapshot (11) with Features 7-1 and 7-2 Extracted.

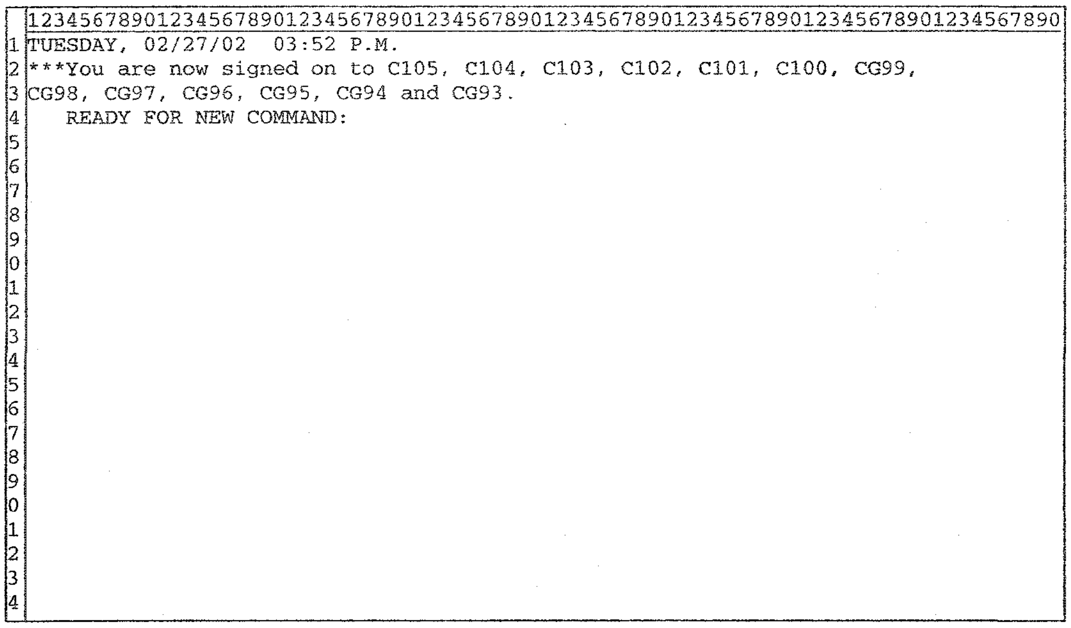


```
 1234567890123456789012345678901234567890123456789012345678901234567890123456789
1                          FEDERAL LEGISLATION
2
3 These files track and describe legislation (bills and resolutions) introduced
4 in the US Congress, from 1973 (93rd Congress) through 1998 (105th Congress).
5 Each file covers a separate Congress.
6
7   CHOICE                                                         FILE
8   1   Congress, 1981-82          (97th)                          CG97
9   2   Congress, 1983-84          (98th)                          CG98
0   3   Congress, 1985-86          (99th)                          CG99
1   4   Congress, 1987-88          (100th)                         C100
2   5   Congress, 1989-90          (101st)                         C101
3   6   Congress, 1991-92          (102nd)                         C102
4   7   Congress, 1993-94          (103rd)                         C103
5   8   Congress, 1995-96          (104th)                         C104
6   9   Congress, 1997-98          (105th)                         C105
7   The 106th Congress, 1999-2000, can be found at:  http://thomas.loc.gov/
8
9  11   Search all Congresses on LOCIS 1973-1998
0       Earlier Congresses:  press ENTER
1  12   Return to LOCIS MENU screen
2
3       Choice:
4                                                                LEGISLATION1
```

Table Start Column	3
Table Start Row	7
Table Width	71
Table Height	9
Number of Columns	5

Feature 7-1	T
Feature 7-2	3_7_71_9_5_

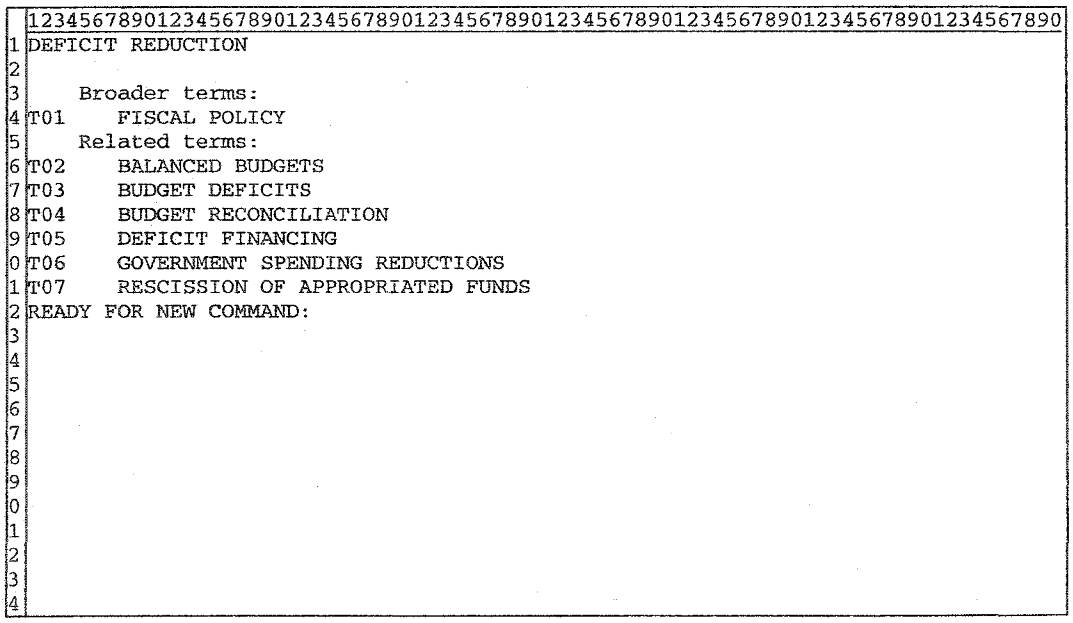
Figure 4.21. An Example Legacy Screen Snapshot (12) with Features 7-1 and 7-2 Extracted.



```
TUESDAY, 02/27/02  03:52 P.M.
***You are now signed on to C105, C104, C103, C102, C101, C100, CG99,
CG98, CG97, CG96, CG95, CG94 and CG93.
   READY FOR NEW COMMAND:
```

Feature 7-1	Blank
Feature 7-2	Blank

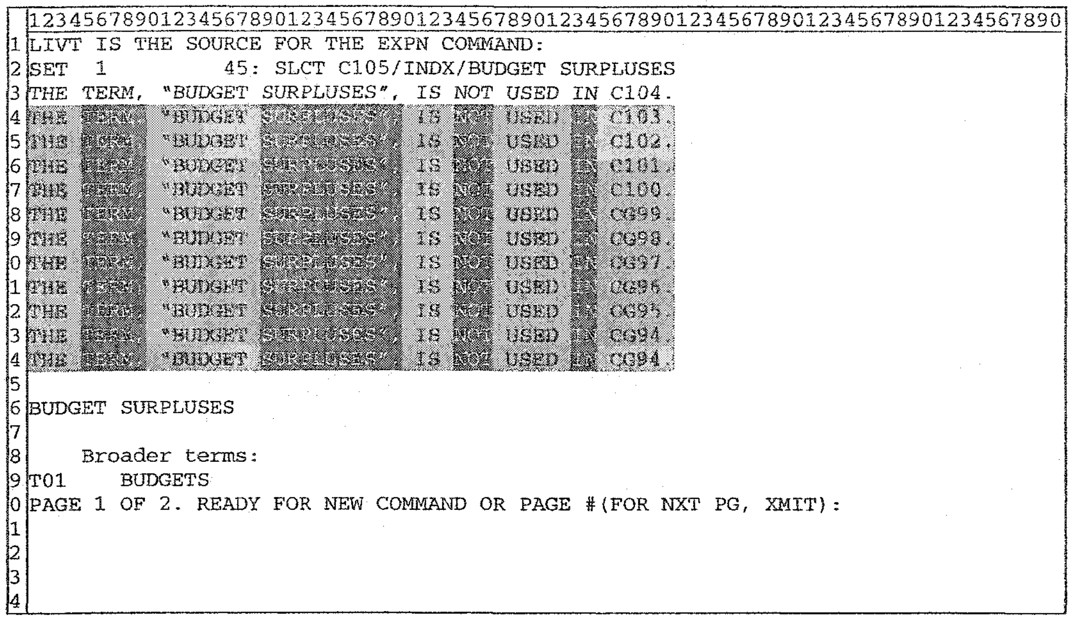
Figure 4.22. An Example Legacy Screen Snapshot (13) with Features 7-1 and 7-2 Extracted.



```
DEFICIT REDUCTION

    Broader terms:
T01    FISCAL POLICY
    Related terms:
T02    BALANCED BUDGETS
T03    BUDGET DEFICITS
T04    BUDGET RECONCILIATION
T05    DEFICIT FINANCING
T06    GOVERNMENT SPENDING REDUCTIONS
T07    RESCISSION OF APPROPRIATED FUNDS
READY FOR NEW COMMAND:
```

Feature 7-1	Blank
Feature 7-2	Blank

Figure 4.23. An Example Legacy Screen Snapshot (14) with Features 7-1 and 7-2 Extracted.

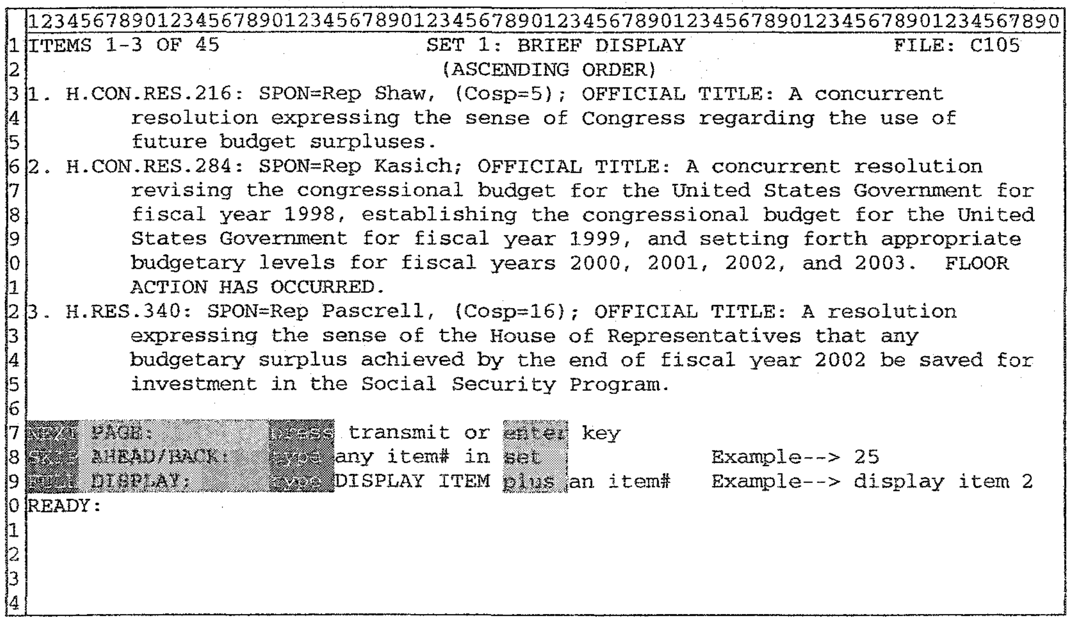


```
 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
1LIVT IS THE SOURCE FOR THE EXPN COMMAND:
2SET  1          45: SLCT C105/INDX/BUDGET SURPLUSES
3THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C104.
4THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C103.
5THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C102.
6THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C101.
7THE TERM, "BUDGET SURPLUSES", IS NOT USED IN C100.
8THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG99.
9THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG98.
0THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG97.
1THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG96.
2THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG95.
3THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG94.
4THE TERM, "BUDGET SURPLUSES", IS NOT USED IN CG94.
5
6BUDGET SURPLUSES
7
8    Broader terms:
9T01    BUDGETS
0PAGE 1 OF 2. READY FOR NEW COMMAND OR PAGE #(FOR NXT PG, XMIT):
1
2
3
4
```

Table Start Column	0
Table Start Row	3
Table Width	50
Table Height	11
Number of Columns	9

Feature 7-1	T
Feature 7-2	0_3_50_11_9_

Figure 4.24. An Example Legacy Screen Snapshot (15) with Features 7-1 and 7-2 Extracted.



```
 1234567890123456789012345678901234567890123456789012345678901234567890123456789012345678901234567890
1ITEMS 1-3 OF 45                SET 1: BRIEF DISPLAY               FILE: C105
2                                (ASCENDING ORDER)
31. H.CON.RES.216: SPON=Rep Shaw, (Cosp=5); OFFICIAL TITLE: A concurrent
4        resolution expressing the sense of Congress regarding the use of
5        future budget surpluses.
62. H.CON.RES.284: SPON=Rep Kasich; OFFICIAL TITLE: A concurrent resolution
7        revising the congressional budget for the United States Government for
8        fiscal year 1998, establishing the congressional budget for the United
9        States Government for fiscal year 1999, and setting forth appropriate
0        budgetary levels for fiscal years 2000, 2001, 2002, and 2003.  FLOOR
1        ACTION HAS OCCURRED.
23. H.RES.340: SPON=Rep Pascrell, (Cosp=16); OFFICIAL TITLE: A resolution
3        expressing the sense of the House of Representatives that any
4        budgetary surplus achieved by the end of fiscal year 2002 be saved for
5        investment in the Social Security Program.
6
7NEXT PAGE:          press transmit or enter key
8SKIP AHEAD/BACK:    type any item# in set            Example--> 25
9FULL DISPLAY:       type DISPLAY ITEM plus an item#   Example--> display item 2
0READY:
1
2
3
4
```

Table Start Column	0
Table Start Row	16
Table Width	42
Table Height	3
Number of Columns	4

Feature 7-1	T
Feature 7-2	0_16_42_3_4_

Figure 4.25. An Example Legacy Screen Snapshot (16) with Features 7-1 and 7-2 Extracted.

4.5 Summary of LeNDI's Discrete Feature Set

Table 4.3 summarizes LeNDI's feature suite, described in sections 4.2 to 4.4. The column "comparison" describes what similarity measure is used to compare two values of the same feature. B means binary comparison, whose output is 1 or 0, i.e., either the two feature values are identical or not. D means discrete comparison, which means that two values of a multi-part feature are compared part by part. Euclidean similarity measure is used which is the number of similar parts divided by the total number of parts. D2 means that discrete comparison is done and weighted Euclidean similarity is used, i.e., there is a different weight for different types of matches and/or a penalty for mismatches. For example, for features composed of sequences of bits, a matching "1" may be more important than a matching "0".

Feature	Description	Comparison
1-1	It is an encoding of the classification of the content of 8 important areas at the periphery of the snapshot to empty, code, title, date, time, page number or message area.	D
1-2	A concatenation of the start columns of all title and code areas discovered, ordered from area 1 to 8.	B
1-3	The text on one of the key 8 areas selected by the user or automatically by LeNDI. Numbers are replaced by "!"s	B
1-4	Similar to 1-3, but with another area chosen.	B
5-1	The label to the left of the initial cursor location. Numbers are replaced by "!"s.	B
2-1	A hash function of the number of IBM 3270 fields and their locations	B
2-2	The number of IBM 3270 unprotected fields received with the snapshot	B
6-1	All characters binary vertical profile	D
6-2	All characters binary horizontal profile	D
6-3	Numbers binary vertical profile	D2
6-4	Words horizontal profile	D
6-5	Special characters binary profile	D2
7-1	An encoding of the snapshot layout classification to "Table", "List" or "General"	B
7-2	Multi-part specifications of the layout classification encoded in Feature 7-1, if it is a "Table" or "Label"	D

Table 4.3. A Summary of the Discrete Feature Suite of LeNDI.

4.6 LeNDI's Binary Feature Set

The three feature subsets described in subsections 4.2 to 4.4 are the primary source for extracting a binary feature set that is used by LeNDI's top-down clustering algorithm [EISSM01], which is described in chapter 5. The top-down clustering algorithm needs binary feature because it produces a binary decision tree. Each leaf of this tree represents a cluster of similar snapshots and each branching node represents a decision to split a group of snapshots to two groups based on a feature-value combination, as explained in details in chapter 5. This means that the snapshots that share this value for this feature are grouped together and those who do not are grouped together. Thus, the comparison of the feature values should be binary, i.e., it should give either a one or a zero.

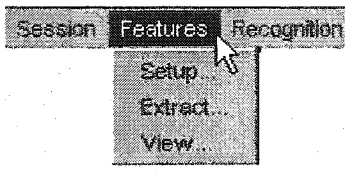
To extract the binary features, multi-part discrete features are either broken down to a number of binary features or abstracted by a number of binary features. Table 4.4 summarizes LeNDI's binary feature suite. Feature 600 abstracts the "all characters binary vertical profile" by dividing it into four equal sequences of bits. Each sequence is represented by half or more of its bits are 1s and is represented by 0 otherwise. The resulting 4 bits are stored as a hexadecimal number. The same is done to extract Feature 601 from Feature 6-2. Feature 602 is an abstraction of Feature 6-3, derived in a similar way with the exception that if any 1 exists in the bit sequence then the sequence is represented by 1. Similarly, Feature 608 is extracted from Feature 6-5.

Feature	Description
100 to 107	Each of these features is a number from 0 to 6 encoding the classification of one of the eight important areas of a snapshot that were described in section 4.2. So, it is a breakdown of Feature 1-1 of subsection 4.2.2.1. Feature 100 corresponds to area 1. Feature 107 corresponds to area 8
108 to 115	These are the starting column numbers for the eight important areas.
116 to 123	These are the actual text content of areas 1 to 8, with numbers replaced by "!"s.
124	Page number extracted from any of the eight areas that is classified as page number information, if one exists
200	The hash function of the number of IBM 3270 fields and their locations described in section 4.3.1 as Feature 2-1.
201	The number of IBM 3270 unprotected fields received with the snapshot, which is Feature 2-2 of subsection 4.3.2.
500	This is the cursor label described in subsection 4.2.2.4 as Feature 5-1.
600	An Abstraction of Feature 6-1 (All characters Binary Vertical Profile)
601	An Abstraction of Feature 6-2 (All characters Binary Horizontal Profile)
602	An Abstraction of Feature 6-3 (Numbers Binary Vertical Profile)
603 to 606	These four features from 603 to 606 represent the hexadecimal representation of the word count of the top row, the second top row, the second last row and the last row, respectively.
607	An encoding of which special character is used in extracting Feature 6-5.
608	An Abstraction of Feature 6-4 (Special Character Binary Profile)
700	This is the layout classification of the snapshot, same as Feature 7-1.
701	This is a hash encoding of rows, columns and other List or Table features

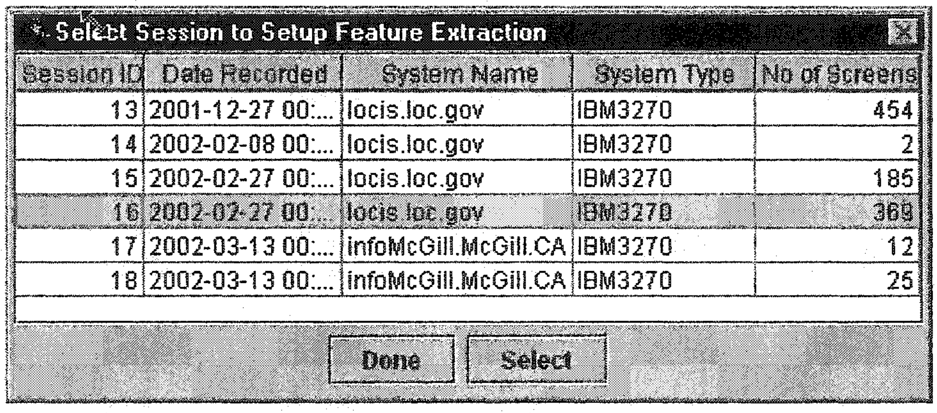
Table 4.4. A Summary of the Binary Feature Suite of LeNDI.

4.7 LeNDI's Feature Extractor and Feature Viewer

Figure 4.26 shows the UI of LeNDI's Feature Extractor module. Figure 4.26(a) shows the menu of LeNDI's Feature Extractor. It allows the analyst to open the feature extracting setup window, to start extracting feature vectors for the snapshots of a trace, and to view the feature vectors extracted for a recorded trace. Since, the Feature Extractor needs some user setup before starting feature extraction, the analyst can choose to keep the default setup values or change them. But first, the analyst needs to choose the recorded traces to work on from the UI shown in Figure 4.26(b). For each recorded trace, Figure 4.26(b) shows the date of recording, the IP of the host, the type of connection used and the number of screen snapshots recorded. After selecting a trace, the tabbed pane of Figure 4.26(c) appears. It allows the analyst to set or change the default keywords and patterns that are used to identify the pieces of data found at the periphery of a snapshot as date, time, page or message. It also, allows the analyst to change the default parameters for projection profile construction and layout classification. Finally, it allows her/him to pick which areas to choose for exact matching, i.e. to take their textual content as Features 1-3 and 1-4, or to tick a check box to let LeNDI pick two areas automatically. LeNDI has a Feature Viewer, whose UI is shown in Figure 4.27. It can be opened by choosing "View.. " from the menu of Figure 4.26(a). In Figure 4.27, the left column shows the trace number, the second left column contains the serial number of the snapshots of the chosen trace. The middle area shows the presentation space of the selected snapshot. The right column shows the feature vector of the select snapshot.



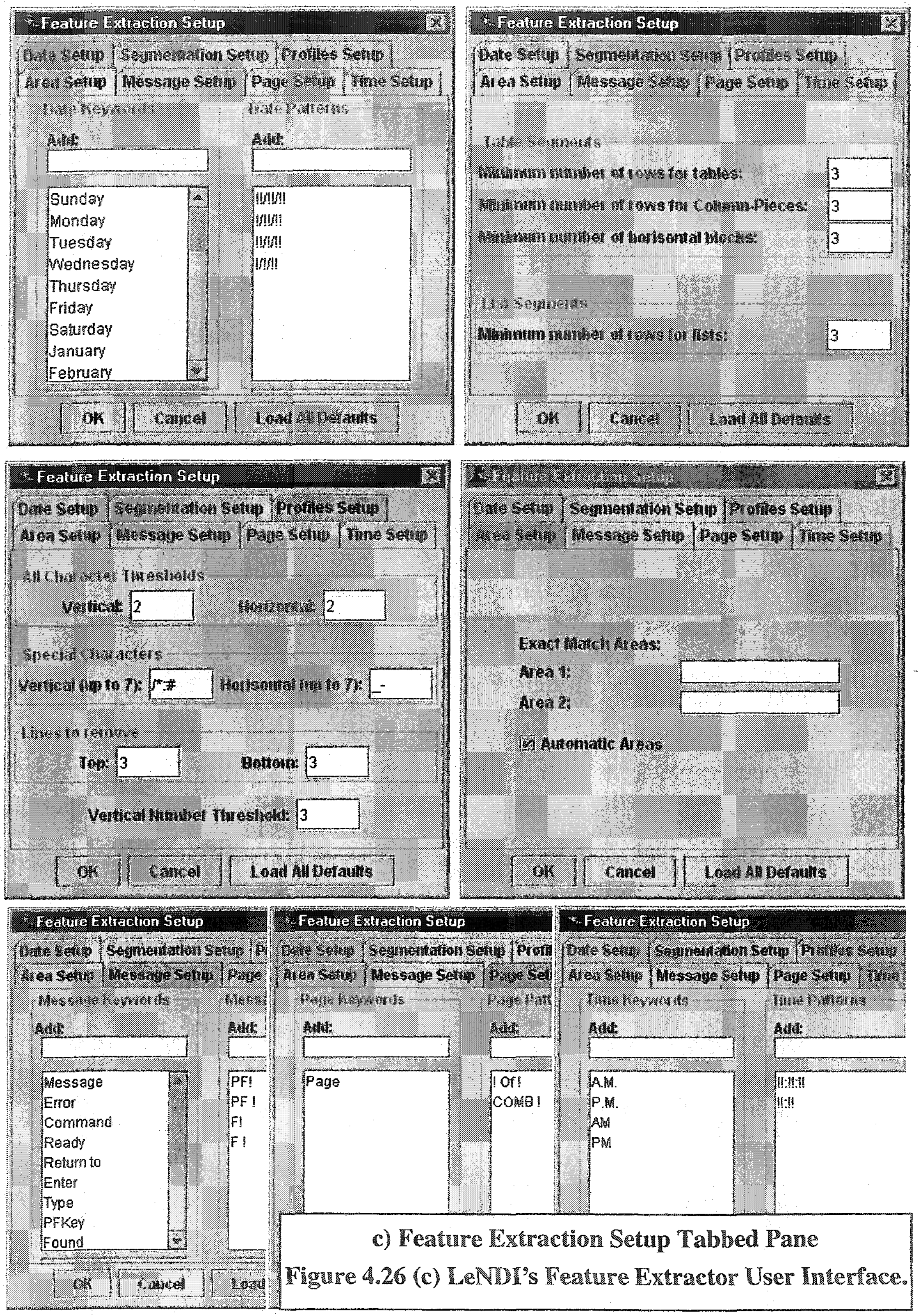
(a) Feature Extraction Menu



Session ID	Date Recorded	System Name	System Type	No of Screens
13	2001-12-27 00:...	locis.loc.gov	IBM3270	454
14	2002-02-08 00:...	locis.loc.gov	IBM3270	2
15	2002-02-27 00:...	locis.loc.gov	IBM3270	185
16	2002-02-27 00:...	locis.loc.gov	IBM3270	369
17	2002-03-13 00:...	infoMcGill.McGill.CA	IBM3270	12
18	2002-03-13 00:...	infoMcGill.McGill.CA	IBM3270	25

(b) Selecting a Recorded Session (Interaction Trace) for Feature Extraction

Figure 4.26. LeNDI's Feature Extractor User Interface.


Feature Extraction Setup
Date Setup | Segmentation Setup | Profiles Setup
Area Setup | Message Setup | Page Setup | Time Setup
OK | Cancel | Load All Defaults

Date Keywords — Add: Sunday, Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, January, February

Date Patterns — Add: !!/!!/!!, !/!!/!!, !!/!/!!, !/!/!!

Table Segments
Minimum number of rows for tables: 3
Minimum number of rows for Column-Pieces: 3
Minimum number of horisontal blocks: 3

List Segments
Minimum number of rows for lists: 3

All Character Thresholds
Vertical: 2 Horizontal: 2

Special Characters
Vertical (up to 7): |*:# Horisontal (up to 7): _-

Lines to remove
Top: 3 Bottom: 3

Vertical Number Threshold: 3

Exact Match Areas:
Area 1:
Area 2:
☑ Automatic Areas

Message Keywords — Add: Message, Error, Command, Ready, Return to, Enter, Type, PFKey, Found

Message Patterns — Add: PF!, PF !, F!, F !

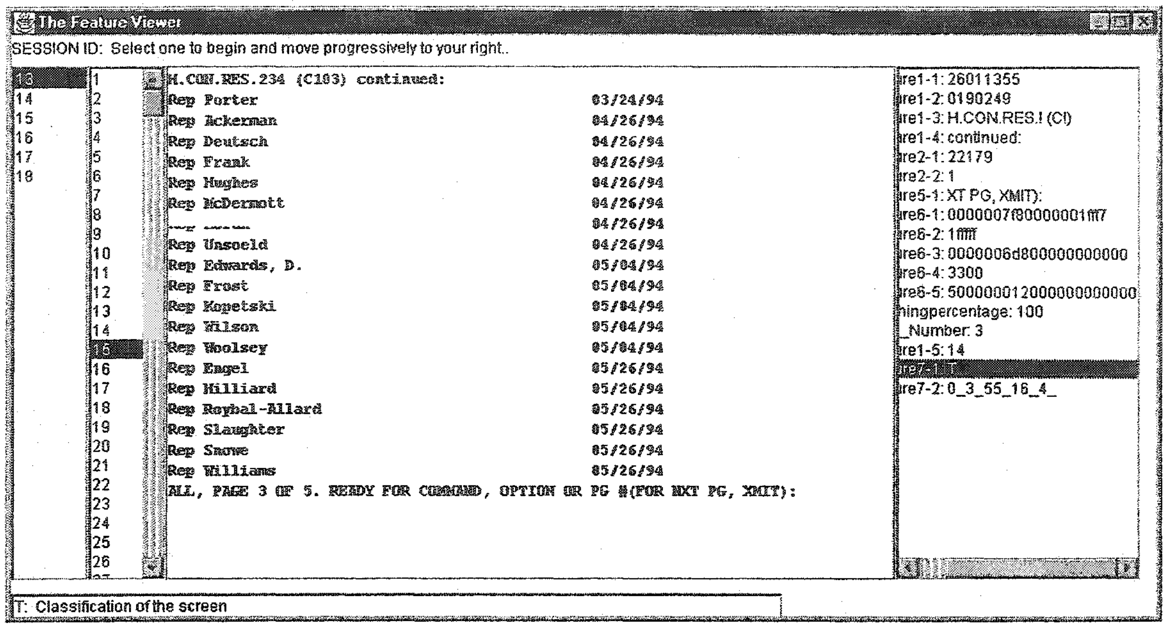
Page Keywords — Add: Page

Page Patterns — Add: ! Of !, COMB !

Time Keywords — Add: A.M., P.M., AM, PM

Time Patterns — Add: !!:!!:!!, !!:!!

c) Feature Extraction Setup Tabbed Pane
Figure 4.26 (c) LeNDI's Feature Extractor User Interface.



SESSION ID: Select one to begin and move progressively to your right..

```
H.CON.RES.234 (C103) continued:
Rep Porter                                      03/24/94
Rep Ackerman                                    04/26/94
Rep Deutsch                                     04/26/94
Rep Frank                                       04/26/94
Rep Hughes                                      04/26/94
Rep McDermott                                   04/26/94
                                                04/26/94
Rep Unsoeld                                     04/26/94
Rep Edwards, D.                                 05/04/94
Rep Frost                                       05/04/94
Rep Kopetski                                    05/04/94
Rep Wilson                                      05/04/94
Rep Woolsey                                     05/04/94
Rep Engel                                       05/26/94
Rep Hilliard                                    05/26/94
Rep Roybal-Allard                               05/26/94
Rep Slaughter                                   05/26/94
Rep Snowe                                       05/26/94
Rep Williams                                    05/26/94
ALL, PAGE 3 OF 5. READY FOR COMMAND, OPTION OR PG #(FOR NXT PG, XMIT):
```

```
re1-1: 26011355
re1-2: 0190249
re1-3: H.CON.RES.I (CI)
re1-4: continued:
re2-1: 22179
re2-2: 1
re5-1: XT PG, XMIT):
re6-1: 0000007f80000001fff7
re6-2: 1ffff
re6-3: 0000006d800000000000
re6-4: 3300
re6-5: 50000001200000000000
ningpercentage: 100
_Number: 3
re1-5: 14
re7-1: T
re7-2: 0_3_55_16_4_
```

T: Classification of the screen

Figure 4.27. LeNDI's Feature Viewer User Interface.

4.8 Discussion and Conclusions

In LeNDI, we implemented and experimented with a new advanced set of snapshot features mainly to serve the automated clustering of similar snapshots together and to minimize the human effort needed to guide the clustering process. Hence, the inference of a unique predicate for each cluster based on this feature set is automated. Such a predicate captures the commonality between the snapshots of one cluster and distinguishes them from those belonging to other clusters. It is to be used at runtime for identifying incoming new snapshots by classifying them to one of the clusters or CUI states already identified. In the current manual practices, an analyst keeps going through many snapshots of the same screen online, and possibly offline, trying to infer their commonality. The analyst formulates this community in the form of a predicate or signature. As described in subsection 2.2.3, some tools offer rich text pattern languages to empower the analyst in his search for such predicate. But these languages mainly rely on finding some keywords existing (or missing) at fixed locations or within some ranges on the snapshots of the screen under analysis. Thus, these languages mainly utilize the snapshot text to find such predicates. Mostly, they do not look into the snapshot content, layout and organization, the semantics of the content or the other invisible information received with IBM 3270 data streams.

In order to be equipped to reverse engineer a wide variety of legacy CUI styles, LeNDI utilizes a broad set of features. It has a subset of features extracted from the special information that may exist at the screen snapshot periphery. It has a second subset of features extracted from the non-visual information received with the outbound data stream. Its third subset is extracted by analyzing the snapshot layout and organization. The division of LeNDI's features suite to three logical subsets, led to thinking of a screen as formed of different layers. A legacy screen can be comprehended at different levels corresponding to these layers. The following main levels of understanding or layers can be identified for a legacy screen snapshot, although the boundaries between them are not well defined:

1. Lexical/Syntactic layer. This layer describes the types of different elements of the screen and the order and location of the visual elements. The important element types that can be used as features are screen title, code, date, time, command line, messages and IBM 3270 data fields and character and field attributes.
2. Layout layer. This layer is a description of the density and distribution of the screen content on its presentation space, the different components of this content and their spatial relation to one another.
3. Semantics layer. This layer includes the meanings of the different components of the screen content. Combined together, these meanings define the function of the screen.
4. Navigation layer. This layer encompasses the navigation sequence (screens and user actions) followed to reach the current screen and the user actions permissible on it.

Depending on the data transfer protocol and the CUI style used to design the screens, legacy CUI screens can vary a lot in terms of the information available in each layer. The interaction reverse engineering process of LeNDI went beyond the comparisons of simple texts on a screen snapshot, to advanced syntax and layout analysis of the snapshot using heuristics and document analysis methods to infer some of the snapshot characteristics. Also, it utilizes the hidden information received with the IBM 3270 data stream to deduce a few features.

However, there is room for improvement. Our future research to enhancement the interaction reverse engineering process will include analysis of the semantic and navigation layers of screen snapshots to empower LeNDI with additional feature subsets.

For example, by treating the set of available screens snapshot as a set of documents and applying information retrieval methods to analyze their content, one can automatically retrieve a set of index terms for the snapshot set. Then, clustering similar snapshots becomes like grouping similar documents together based on their content represented by index terms. Another example is adding a navigation history segment of chosen length as a feature for every snapshot. This can be the immediate predecessor snapshot and the action done to move to the current snapshot.

Chapter Five

Legacy User Interface Behavior Modeling

In chapter 4, the necessary steps for automating the process of building the state-transition model of a legacy system CUI were introduced:

1. Extracting a rich set of features for every snapshot in the recorded traces, automatically,
2. Defining a similarity measure for each feature,
3. Defining a similarity and/or distance function to use for clustering similar snapshots together,
4. Clustering similar snapshots together separate from the rest,
5. Verifying and correcting the clustering results via user feedback
6. Automatically extracting unique predicates that distinguish the snapshots belonging to different clusters, i.e., to different legacy screens, and
7. Modeling the permissible user behavior (actions) on every legacy screen

Chapter 4 covered steps 1 and 2 above. It presented the set of features that are extracted to every recorded screen snapshot, the possible user setup to control feature extraction and the similarity measures used to compare two values of each feature. Then, at the end of feature extraction, every recorded snapshot is represented by a feature vector. These vectors along with feature similarity metrics are the input to the actual process of legacy CUI behavior modeling. This process covers steps 3 to 7 above and is implemented by tasks T1.2 to T1.4 in Figure 3.1. This chapter gives the full details of legacy CUI behavior modeling in LeNDI. The output of this process is a state-transition model of the legacy CUI.

5.1 Introduction

Like mapping the streets of an area, legacy CUI behavior modeling captures the roads (actions) and intersections (screens) of a legacy CUI. The produced state-transition model is the "map" of the corresponding CUI. The model serves the following purposes:

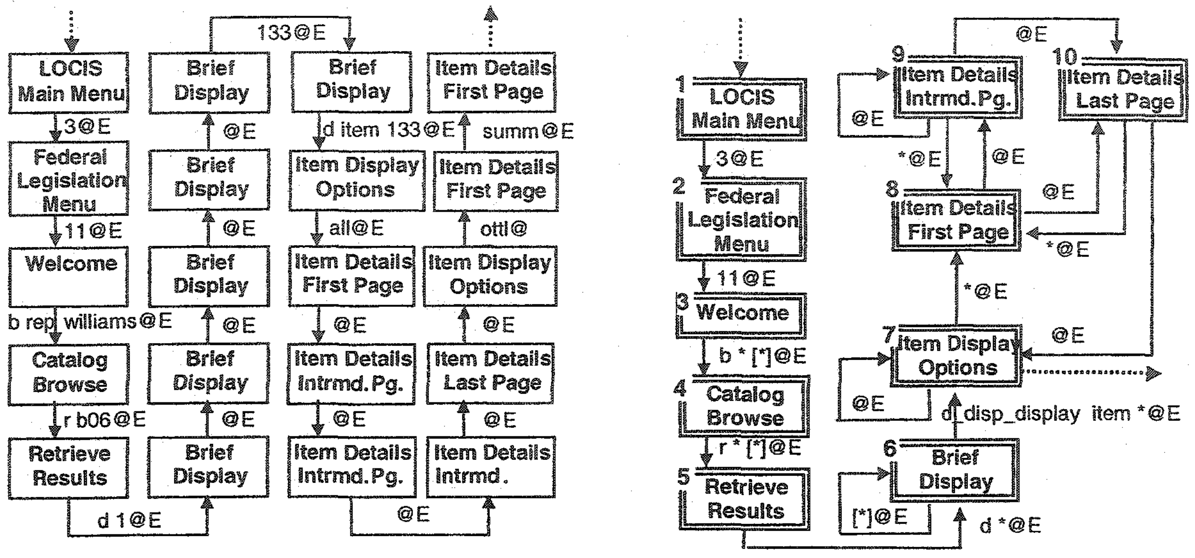
1. First, the state-transition model is a documentation of the legacy CUI behavior and can be used for understanding the application behavior and capabilities.
2. Second, the main function of the model is to serve as a "guide" for the new re-engineered UI in navigating the legacy CUI. So, when the new UI front-end executes a task in the legacy host and navigates through its screens, it checks the identity of every newly received snapshot at runtime against the states of the model to identify to which screen the new snapshot belongs. Then, the data input and output mandated by the task plan on this legacy UI screen take place followed by the execution of the necessary action to proceed to the next state.

A state-transition model of a legacy CUI does not need to cover the entire CUI. It may only cover the parts that will be subject to reengineering. The rest of this introduction includes three subsections. Subsection 5.1.1 gives an example state-transition model. Subsection 5.1.2 presents the state-transition modeling problem. Subsection 5.1.3 outlines the solution implemented in LeNDI to solve this problem, which is detailed in the rest of this chapter.

5.1.1 Example

Traditionally, state-transition models have been used to specify the dialog between the user and the application through the user interface, for the purposes of model-based interface development and evaluation [Sch99]. Figure 5.1 shows a schematic diagram of a segment of an interaction trace with the Library of Congress Information System (LOCIS) [LOCIS], through its public IBM 3270 connection, and the corresponding portion of the state-transition model of LOCIS CUI. LOCIS is a command-driven legacy library information system that allows performing information retrieval tasks on Braille and Audio and Federal Legislation catalogs of the Library of Congress. The interaction trace segment of Figure 5.1(a) started by accessing LOCIS main menu and then the Federal Legislation menu. Then the user selected the catalog he wanted to open and got a welcome screen snapshot. By issuing the right commands, he browsed the catalog, retrieved a subset of its entries, displayed it briefly and finally, displayed the details of the entry he was looking for.

@E	Enter Key	[*]	Optional Argument
*	Mandatory Argument	x_y_z	x, y and z are alternatives of the same keyword



(a) A segment of an interaction trace with LOCIS.

(b) The part of the state-transition graph corresponding to the segment in (a).

Figure 5.1. An Example Trace of User Interaction with the Library of Congress Information System (LOCIS) and the Corresponding State-Transition Model.

5.1.2 Problem Formulation and Definitions

As a directed graph, a state-transition model of a legacy CUI, inferred from a recorded trace of interaction with the legacy UI, can be defined as follows:

Definition 5.1

$UI_{model} = (States_{UI}, Transitions_{UI})$

1. $States_{UI} = \{St_i, i=1\ldots\#\text{states}\}$,
   $snap \ni Trace_{j,n} \wedge Trace_{j,n} \ni TraceSet \Rightarrow \exists\, St \ni States_{UI} \wedge$ instance-of $(snap, St)$,
2. $Transitions_{UI} = \{\ (St_{source}, St_{destination})\ \}$,
   $(St_l, St_m) \ni Transitions_{UI} \Rightarrow \exists\ (snap_{j-1}, key_j, snap_j) \ni Trace_{j,n} \wedge Trace_{j,n} \ni TraceSet$
   $\wedge$ instance-of $(snap_{j-1}, St_{source}) \wedge$ instance-of$(snap_j, St_{destination})$
3. $TraceSet = Trace_{1,n_1}\ (Trace_{j,n_j})^*\ \ j=2\ldots m$
   - where $Trace_{j,n_j}$ is a trace of length is n_j, as defined in Definition 3.1.

According to Definition 5.1, each snapshot in the recorded trace is an instance of a state in the legacy CUI state-transition model. Furthermore, for each transition in the model there must exist at least one keystroke sequence in the recorded trace that leads from a snapshot, which is an instance of the source state, to another snapshot, which is an instance of the destination state. In this work, the terms "legacy screen" or simply "screen", "state", "node", and "cluster" refer to the same thing from different views, and hence will be used interchangeably depending on the context. A legacy screen refers to a legacy CUI unit, represented by a matrix of characters and other associated information which exhibits certain behavior in terms of the user actions it permits and the outcome (destination screen) of each action. This screen reflects a state of the legacy application CUI, represented by a node on the state transition graph. Such a state is represented by a predicate that is inferred by clustering the recorded instances or snapshots of the corresponding screen in one cluster.

Building the state-transition model of a legacy CUI can be divided to three sub-problems. The first problem is identifying the distinct behavioral states of the legacy CUI, represented by nodes on the model. To do so, using interaction traces as the only input, one needs to identify in these traces the snapshots that are similar to each other, according to some suitable similarity measure, and exhibit identical behavior. These similar snapshots should be instances of the same state. Identifying them is the first step in modeling this state.

Then, the second problem is: given the similar snapshots of every state, how to infer the common identity of these snapshots represented by some unique predicate? This predicate is needed to recognize new snapshots as instances of existing states. In other words, the problem here is how to induce a classifier that is able to classify new snapshots received at runtime as instances of an existing state?

The third problem is building models of the transitions permitted at each state, i.e., the user behavior associated with each node. These transitions are the arcs of the state-transition model. A transition model captures the commonality of all action instances recorded that caused the corresponding transition, e.g., command keywords, actions locations, etc. The next subsection outlines LeNDI's solution to these three problems, before providing full details in the next sections.

5.1.3 LeNDI's Approach to Legacy CUI Modeling

LeNDI adopts clustering as the solution to the first problem of identifying the distinguished behavioral states of a legacy CUI. In this solution, "similar" snapshots are clustered together in one cluster. A similarity function is used to decide which snapshots are similar. This function utilizes the features described in chapter 4. These features are extracted from the visual syntax and semantic characteristics of the screen snapshot and from the other information provided by the data transfer protocol with each snapshot.

Clustering is a generic problem with instances in a variety of application domains. Clustering algorithms are so many to the extent that you find popular sayings like "There are more clustering techniques suggested than the number of real-world problems solved with them" or "Clustering algorithms are worth a dime a dozen" [Mir96]. In general, clustering algorithms are either batch, assuming that the complete set of input instances is available at the same time, or incremental, allowing for additional instances to be provided after initial clustering. Incremental algorithms, given a new instance, decide the cluster to which it belongs by evaluating how similar the new instance is to the existing clusters. Batch clustering algorithms are either top-down or bottom-up or hybrid. Top-down algorithms start with a single cluster and continuously decompose it until a stopping criterion is met. Bottom-up algorithms start with each instance belonging to a cluster by itself and join clusters until a stopping criterion is met. Irrespective of their control flow, all clustering algorithms require a distance (or similarity) metric, on the basis of which, it is decided whether to split a cluster (in top-down algorithms), or whether to join two clusters (in bottom-up algorithms), or whether a new instance is similar enough to any of the existing clusters (in incremental algorithms). Any such metric depends on a set of features that describe the input instances. In our case, these are the feature vectors extracted for every snapshot as described in chapter 4.

Two clustering algorithms have been implemented in LeNDI: an incremental algorithm [SEKSM99] and a top-down algorithm that stops when the number of expected clusters is reached [EISSM01]. These two algorithms have different knowledge requirements and each one is preferable under different usage scenarios. The incremental clustering algorithm uses the discrete (original) feature set of LeNDI described in sections 4.2 to 4.5 and summarized in Table 4.3. It requires reasonable knowledge of the legacy system under analysis and making decisions such as which features would be more likely to be effective and what weights to assign to them. On the other hand, the top-down algorithm uses LeNDI's binary feature set (section 4.6 and Table 4.4) and requires almost no input from the user other than an initial estimate of the number of expected clusters. Thus, each algorithm might be more suitable for certain application problems. Also, we have found that, in practice, it is useful in some cases to explore the traces of a legacy system thoroughly with the single-path incremental algorithm until getting an accurate estimate of the number of clusters, and then using the more automated top-down algorithm to generate an almost correct partition of the trace snapshots. A partition is a set of non-overlapping clusters such that each recorded snapshot belongs to only one cluster. For example, {{1,2}, {3}}, {{1}, {2}, {3}} and {{1}, {2,3}} are three different partitions of the trace {1,2,3}. A number of bottom-up clustering algorithms were explored while developing LeNDI, but none of them gave satisfactory results.

The result of clustering is a partition of the entire snapshot set. Due to the diverse and unpredictable nature of legacy snapshot data, it is likely that a number of clustering rounds would be needed for a given data set. So, the LeNDI analyst would review the results of each round of clustering, readjust whatever clustering parameters needed by the clustering algorithm s/he is using, and then re-cluster the data. Additionally, both algorithms allow user feedback to fix clustering errors by merging or dividing clusters or moving snapshots from a cluster to another.

LeNDI employs classifier induction to solve the second problem of capturing a common identity predicate for each CUI state, represented by a cluster of snapshots. So, once a correct partition has been produced, the LeNDI analyst can induce a classifier that can correctly classify the individual snapshots into their corresponding clusters. LeNDI implements two different classifiers. The first is a simple signature-based classifier that captures the commonality of the snapshots of every cluster in a predicate. The second is a decision tree classifier, which extends the decision tree produced by the top-down clustering algorithm according to the user feedback on the partition produced by clustering. After building a classifier, it can then be used at runtime to recognize new, previously unseen snapshots as instances of the legacy CUI states.

To solve the third problem of identifying the arcs of the state-transition model of a legacy CUI, LeNDI develops a model of each possible transition that had some instance(s) recorded in the interaction traces. The only input to this process is samples of the user actions done to perform such a transition along with the locations on the snapshots where they occurred. These samples need to be grouped and analyzed to infer their commonality and variability and formulate this information in a model. Furthermore, location information of action instances can be analyzed to infer the fixed location or range of locations within which the action takes place on its origin screen. The generality of the model produced for every action depends on the number of instances of this action available in the recorded interaction traces.

The rest of this chapter is organized as follows: Section 5.2 describes snapshot clustering process in LeNDI and its two clustering algorithms, the single-path incremental algorithm and the top-down algorithm. Section 5.3 describes LeNDI's two *classifier induction methods, the signature-based classifier and the decision tree classifier.* Section 5.4 presents transition modeling in LeNDI. Section 5.5 is an evaluation of the legacy CUI behavior modeling process. Section 5.6 concludes with a discussion of the overall process, its strengths and limitations and possible extensions.

5.2 Clustering Legacy Screen Snapshots in LeNDI

This section presents the process of clustering similar snapshots in LeNDI as instances of the same CUI state. It starts with a detailed description of the two clustering algorithm implemented in LeNDI, the single-path incremental algorithm and the top-down algorithm. Then, it follows by a description of the clustering quality metric employed in LeNDI, MoJo Plus. Finally, it concludes by the describing QandA, CelLEST visualization tool that allows reviewing and revising clustering results.

5.2.1 Clustering Method 1: Single-path Incremental Clustering of Legacy Snapshots

The single-path incremental clustering algorithm deployed in LeNDI is derived from a generic version described in chapter 3 of "*Information Retrieval*" by van Rijsbergen [Rij79]. The algorithm views clusters as centered at a representative point, the *centroid.* Cluster representatives or centroids can be calculated in different ways. The specifics of cluster representative calculation in LeNDI follow shortly. The algorithm needs the LeNDI analyst to provide a similarity function, i.e., a function that defines how the similarity of two snapshots (or a snapshot and a centroid) can be calculated in terms of their feature vector similarity. Snapshots are accessed sequentially, one after another. Each new snapshot is clustered using the information available so far about the data set, i.e., the feature vectors of the snapshots processed so far. A new snapshot is introduced to the algorithm in the form of a feature vector and is compared to the centroid of each existing cluster. Then it is assigned to the most similar cluster centroid, if its similarity to this centroid is above a user-defined threshold; otherwise it is the first member of a new cluster. This process can be summarized in the following steps:

1. Legacy screen snapshots are processed sequentially;
2. The first snapshot becomes the cluster representative of the first cluster;
3. Each subsequent snapshot is matched against all cluster representatives (centroids) existing at the time using some similarity measure ;
4. A given snapshot is assigned to the cluster whose representative is most similar to it if similarity exceeds or equals a certain threshold;
5. When a snapshot is assigned to a cluster, the centroid of that cluster is recomputed;
6. If the highest similarity of the snapshot with a cluster representative is below the threshold, then the snapshot becomes the cluster representative of a new cluster.

This algorithm is "incremental" because it accesses and clusters the snapshots, of the input data set, one at time using the clusters available so far. It is called "single-path" because it goes over the data set one time only.

In order to measure similarity, LeNDI employs a set of *recognizers*. Each of them is configured to use one or more of the features described in chapter 4. Each feature is assigned a weight. A recognizer compares two snapshots (or a snapshot and a cluster centroid) in terms of its features. It measures the similarity of the two snapshots using each of its features separately. Then, the recognizer's vote is the weighted-sum of the similarity of individual features. The final vote of the entire set is the weighted-sum of the individual votes of its recognizers.

A configuration step is required to set up the recognizers and to define the similarity threshold. The LeNDI analyst needs to review the available traces to judge what features might be more effective in clustering the snapshots of a given system properly. Then s/he needs to decide the number of recognizers to use, the relative weights of their decisions, the features employed by each recognizer and the relative weights of these features. This effort comes with the reward of not having to decide beforehand the number of clusters needed, as is the case with the top-down clustering algorithm explained in subsection

5.2.2. If a feature is missing on a particular screen snapshot, any recognizer that utilizes this feature may be configured to either ignore it, or to consider its absence as evidence that there exists a distinct screen that lacks the feature in question.

Formally, similarity is measured using a set of recognizers R as defined in Definition 5.2. Definition 5.2 shows that the vote of R on the similarity of two feature vectors is in fact a linear combination of the votes of all the individual features employed in all its recognizers. Thus, *Vote R* can be simplified to a linear weighted-sum of the all the features used in all recognizers. But, using recognizers allows using non-linear similarity functions in some recognizers. An example of such non-linearity is feature dependency; i.e., if the values of feature $F_{i,j}$ for $snap_1$ and $snap_2$ are not similar, then ignore comparing feature $F_{i,j+1}$. For example, features $F_{i,j}$ and $F_{i,j+1}$ can be Feature 7-1, the snapshot layout classification and Feature 7-2, the layout specifications. If two snapshots have different layouts, then comparing their specifications is meaningless.

Definition 5.2

A set of recognizers $R = (r_i, i = 1 \ldots \#\text{recognizers})$ is defined such that

1. $r_i = \{F_i, W_i, w_i\}$
   - F_i is vector of the features utilized in r_i. The j^{th} feature is referred to as $F_i\,[j]$, or simply $F_{i,j}$
   - W_i is weight vector of features used in r_i, where $W_i\,[j]$ (or simply $W_{i,j}$) is the weight of feature $F_{i,j}$
   - w_i is the weight of the vote of r_i
2. $Vote\ r_i\,(snap_1, snap_2) = \sum_{j=1..|Fi|} \{\text{ Similarity } (snap_1, snap_2, F_{i,j}) * W_{i,j}\ \}$
   - Similarity ($snap_1$, $snap_2$, $F_{i,j}$) is a function that returns the similarity of two given snapshots (or a snapshot and a centroid), $snap_1$ and $snap_2$, based on comparing their values of feature $F_{i,j}$
3. $Vote\ R\ (snap_1, snap_2) = \sum_{i=1..\#\text{ recognizers}} \{Vote\ r_i * w_i\}$

Algorithm 5.1 is the pseudocode of the single-path incremental algorithm. It takes as inputs a set of recognizers, a similarity threshold and a set of recorded traces of interaction with a legacy system. It outputs a partition P of the input snapshots. Step 1 initializes P to an empty set. Steps 2 and 3 iterate over every snapshot in every recorded trace. For each snapshot $snap_j$, steps 4 and 5 initialize two variables to zero, which will store the maximum similarity vote for the snapshot, *max Vote*, and the Id if the most similar cluster, *most Similar*. Step 6 iterates over every existing cluster. Step 7 measures the similarity of $snap_j$ with the current cluster centroid. If the current cluster is more similar to $snap_j$ than any previously examined cluster, then the similarity value and cluster Id are recorded in steps 8 to 10. After measuring similarity with all clusters, step 11 checks if the maximum similarity is below the threshold *Thresh*. If yes, steps 12 to 15 create a new cluster with only one item for the time being, which is $snap_j$, and adds it to the partition *P*. $snap_j$ is the centroid of this cluster since it is its only item. If the maximum similarity is above or equal to *Thresh*, then steps 16 to 18 assign $snap_j$ to the most similar cluster and update the centroid of this cluster.

The user can make one of two choices while configuring LeNDI's clustering process. The first is to use the first item in a cluster as its representative and never change it during the whole process. In this case, no update takes place. Or s/he can choose to build a centroid and re-calculate every time a new snapshot is added to the cluster. In this case, LeNDI follows a simple procedure to calculate the centroid, which is choosing the mode

```
Algorithm 5.1: Single-path Incremental Clustering
Input: A set of recognizers $R$, a threshold $Thresh$ and a set of traces of snapshots $T$.
Output: A partition $P$ of the snapshots of $T$
Steps:
1. Initialize $P = \varphi$, where $\varphi$ is an empty set
2. For every trace $t_i \in T$, $1 \le i \le |T|$
    3. For every snapshot $snap_j \in t_i$, $1 \le j \le |t_i|$
        4. $max\ Vote = 0$
        5. $most\ Similar = 0$
            6. For every cluster $c_k \in P$
                7. $current\ Vote = Vote\ R\ (snap_j, centroid\ (c_k))$
                8. If ($max\ Vote < current\ Vote$) then
                    9. $max\ Vote = current\ Vote$
                    10. $most\ Similar = k$
        11. If ($max\ Vote < Thresh$) then
            12. Create a new cluster $c_{new}$
            13. Add a $snap_j$ to $c_{new}$
            14. Set $snap_j$ to be $centroid\ (c_{new})$
            15. Add $c_{new}$ to $P$
        16. Else
            17. Add $snap_j$ to $c_{most\ Similar}$
            18. Update $centroid\ (c_{most\ Similar})$
```

Algorithm 5.1. Single-path Incremental Clustering.

(most frequent value) for single-part features (Features 1-2 to 1-4, 2-1, 2-2, 5-1 and 7-2) and the mode of every part for multi-part features (Features 1-1, 6-1 to 6-5 and 7-1). For example if a cluster has three snapshots, with the following values for the multi-part Feature1-1: "025**13**102", "025**03**002" and "025**02**102", the centroid will have the value: "025**03**102", where bold font shows parts that differ among the snapshots and their modes.

After all the snapshots in the recorded traces are clustered, the results can be reviewed using LeNDI's *cluster viewer*, a built-in review and revision module in LeNDI, or using QandA (Questions and Answers) [Vij02], the visualization tool of CelLEST. Based on his/her review, the LeNDI analyst can readjust the similarity measure and re-cluster the snapshot set to achieve the best possible results in his/her judgement.

5.2.2 Clustering Method 2: Top-down Clustering of Legacy Snapshots

This algorithm[4] [EISSM01] was developed to further automate the screen-snapshot clustering process and to relieve the user from having to decide a similarity threshold and feature-weighting schemes, whenever possible. However, this comes with the cost of having to estimate the number of clusters expected in the trace as an input to the algorithm. Subsection 5.5.3 comments on the strengths and weaknesses of both clustering algorithms. The top-down clustering algorithm is the first phase of a two-phase process for legacy CUI state identification. The second phase of this process is a supervised learning phase for classifier induction that modifies the decision tree produced by top-down clustering, according to user feedback. The clustering phase is described here, while the classifier induction phase is described in subsection 5.3.2. This algorithm is a top-down clustering algorithm that starts by putting all the snapshots in one cluster and keeps dividing them into more clusters in a way that minimizes the maximum internal cluster incoherence. It stops, when a user defined criterion is met, which can be the expected number of clusters or a threshold of the maximum incoherence allowed. Using the snapshot binary features of Table 4.4, this algorithm produces a decision tree that can be used to classify the snapshots into screen clusters.

Algorithm 5.2 shows the pseudocode of the top-down clustering algorithm. It takes as input a set of interaction traces T whose snapshots will be clustered, and an estimate of

[4] This algorithm was developed primarily by Paul Iglinski, with help from other CelLEST team members.

the number of screen clusters required and/or the maximum internal cluster incoherence accepted. It outputs a partition of the input snapshots and a decision tree *DT* that defines how this partition was constructed and which feature and value were used for splitting at each tree node. Initially, all the snapshots are placed in a single cluster. The algorithm continues to split one cluster at a time until reaching the desired number of clusters. After many experiments with different data sets and different splitting criteria, we found that the best criterion for splitting a cluster is minimizing the maximal internal cluster incoherence using linear distance averaging. We call this splitting criterion the "best-split test". So, as long as the number of clusters is less than the expected number of clusters, the algorithm identifies the most "incoherent" cluster and splits it into two new clusters, in a way that minimizes the incoherence of the resulting clusters. Incoherence is measured as the average distance of every instance in a cluster from every other instance in the same cluster. Currently, all the features are treated as having discrete non-ordinal values. Each snapshot has 39 features, by default equally weighted at 1. If two instances have different values for a feature, this feature contributes its weight to the distance measure. The distance between two instances is then simply the sum of the weights of their differing features. Currently, LeNDI assigns equal weights to all features.

As an alternative to providing the number of clusters as the determining factor for terminating splitting, the user can specify an incoherence threshold or use a default threshold. Thus, when all clusters are below the specified threshold, the splitting stops. The default threshold value varies in proportion to the sum of all the feature weights.

```
Algorithm 5.2: Top-down Clustering
Input: The number of desired clusters #clusters and/or the maximum cluster incoherence
        threshold incoherence Thresh and a set of interaction traces T
Output: A partition P of the snapshots of T, and a decision tree DT

1. Create a new cluster $c_0$
2. Add all the snapshots in T to $c_0$
3. Initialize P with only one cluster, $c_0$
4. Create an decision tree DT with one root node representing $c_0$
5. While (num Clusters < #clusters OR P.max Incoherence () > incoherence Thresh )
    6. Let $c_{most}$ = Most Incoherent Cluster in P
    7. Create new Split bestSplit = NULL
        8. For each feature $f \in$ LeNDI's Binary Feature Set
            9. Create a set V of all the values of f in $c_{most}$
            10. For each value $v \in V$
                11. Create new Split newSplit = new Split (f,v)
                12. Create clusters $c_{with}$ , $c_{without}$
                13. For each snapshot snap in $c_{most}$
                    14. If snap.featureValue(f) = v then
                        15. Copy snap to $c_{with}$
                    16. Else
                        17. Copy snap to $c_{without}$
            18. If ($c_{most}$.maxIncoherence(newSplit) < $c_{most}$.maxIncoherence(bestSplit) )
                then
                    19. bestSplit = newSplit;
    20. Divide $c_{most}$ to $c_{with}$ and $c_{without}$ according to bestSplit
    21. Remove $c_{most}$ form P
    22. Add $c_{with}$ and $c_{without}$ to P
    23. Replace the leaf node of $c_{most}$ in DT with a decision node that implements
        bestSplit
```

Algorithm 5.2. Top-down Clustering.

Algorithm 5.2 starts by steps 1 to 3 which create a partition *P* with one cluster c_0 that contains all the input snapshots. Step 4 creates a decision tree *DT* with one node that corresponds to c_0. Step 5 iterates over *P* while the termination criterion is not met. This criterion can include one or both of the following: the required number of clusters or a threshold for the maximum internal cluster incoherence permitted. Step 6 picks the most incoherent cluster in *P*, which is c_{most}. Step 7 creates an empty Split *bestSplit*, which is a data structure for storing a feature and a value for this feature as a criterion for splitting a cluster. Steps 8 to 10 iterate over every possible value *v* that exists in any of the snapshots of c_{most} for each feature *f* of the 39 features used. Step 11 creates a new Split *newSplit* to store the current iterator, i.e., feature and value combination. Steps 12 to 17 create two clusters c_{with} and $c_{without}$ and stores in c_{with} copies of the snapshots that have value *v* for feature *f* and stores in $c_{without}$ copies of the snapshot that lack value *v* for feature *f*. Steps 18 and 19 store the current split in *bestSplit* if it produces less maximum internal cluster incoherence than *bestSplit*, if used to split c_{most}. After trying all the possible splits of c_{most} and finding the best-split, steps 20 to 22 divide c_{most} according to *bestSplit* and replaces it in *P* by c_{with} and $c_{without}$. Step 23 adds a new node for the best-split in the decision tree *DT*.

This algorithm has two modes: an automated mode and an interactive mode. In the automated mode, if the LeNDI analyst can estimate the number of unique CUI screens expected for the system under analysis, the algorithm can proceed unsupervised to decompose the original set of snapshots into the expected number of clusters. When it is not possible to give an estimate, or when the analyst prefers to have more control on the algorithm, we have developed an interactive version of the algorithm that can be monitored and guided by the user. The user may step through the algorithm's split decisions. If, at some point, a cluster looks close to its desired state, i.e., it contains no or very few snapshots that do not belong there, the analyst can finalize it, ensuring that the algorithm does not consider it as a candidate for further splitting. Moreover, the analyst can force the algorithm to split a certain cluster in the next step.

An example decision tree produced by Algorithm 5.2 is shown in Figure 5.2. The feature numbers shown, i.e., 103, 114, 102, etc. are the numbers given to the binary features of LeNDI in Table 4.4. C0, C1, etc. are the clusters created by Algorithm 5.2. In this tree, the top node, which corresponds to the first best-split, is based on if feature 103 is 0 or no. If yes, then next node splits the data according to if feature 114 equals 0 or no, and so on and so forth.

```
if feature 103 == 0
+-----if feature 114 == 0
|     +-----if feature 104 == 0
|     |     +-----C3
|     |     else
|     |     +-----C4
|     else
|     +-----if feature 106 == 0
|           +-----C5
|           else
|           +-----C6
else
+-----if feature 102 == 1
      +-----if feature 607 == 7
      |     +-----C0
      |     else
      |     +-----C1
      else
      +-----if feature 124 == 2
            +-----if feature 100 == 2
            |     +-----C9
            |     else
            |     +-----C10
            else
            +-----if feature 100 == 2
                  +-----if feature 102 == 0
                  |     +-----if feature 116 == "SET  1"
                  |     |     +-----C15
                  |     |     else
                  |     |     +-----C16
                  |     else
                  |     +-----C7
                  else
                  +-----if feature 100 == 0
                        +-----C2
                        else
                        +-----if feature 607 == 7
                              +-----if feature 200 == 21958
                              |     +-----C8
                              |     else
                              |     +-----if feature 102 == 3
                              |           +-----C13
                              |           else
                              |           +-----C14
                              else
                              +-----if feature 100 == 3
                                    +-----C11
                                    else
                                    +-----C12
```

Figure 5.2. An Example Decision Tree Produced by the Top-down Clustering Algorithm 5.2.

5.2.3 Clustering Result Visualization and User Feedback

Associated with clustering, is a process of result visualization and user feedback. For both algorithms, this process is necessary to verify the correctness of the results obtained and to fix any clustering mistakes before inducing a classifier. It is also important to review the results obtained between different iterations if multiple clustering iterations were done, particularity for the incremental clustering algorithm, which is iterative in nature.

LeNDI offers a simplified result visualization module that allows reviewing and revising clustering results. But the primary visualization tool of CelLEST is QandA (Questions AND Answers) system[5] [Vij02] which supports the visualization needs of both LeNDI and Mathaino systems through a user-friendly GUI. Using QandA, the LeNDI analyst can inspect the results of clustering a set of input traces, and hence, decide to reconfigure the clustering process and repeat it, in the case of signal-path incremental clustering. In the case of top-down clustering, s/he can change the desired number of clusters or the maximum internal cluster incoherence accepted or switch to the interactive mode where s/he has more control over the clustering process. After these revisions, and when satisfactory results are obtained, the analyst can move any outlier snapshots to their proper clusters or join any redundant clusters to their originals. Then, s/he can ask LeNDI to generate a signature-based or a decision tree classifier for the final partition as described in section 5.3. Figures 5.3 and 5.4 show some snapshots of QandA user interface. Figure 5.3 shows the cluster view of QandA with clusters represented as thumbnails. If a cluster thumbnail is clicked, the thumbnails of its centroid and snapshots are shown in the upper panel. If a cluster thumbnail is double clicked, its centroid is enlarged in a separate window. If a centroid or a snapshot thumbnail is clicked, it is enlarged in a new window.

Figure 5.4 shows the snapshot view of QandA. This view shows the centroid and snapshots of every cluster, connected by lines that represent their similarity. The snapshots closer to the centroid are more similar to it than the farther ones. Through this view, the analyst can easily detect outliers and perform cluster revision. S/he can move snapshots from one cluster to another, merge a set of clusters, and/or split a cluster.

[5] QandA system was developed primarily by Vijayan Menon, with help from other CelLEST team members

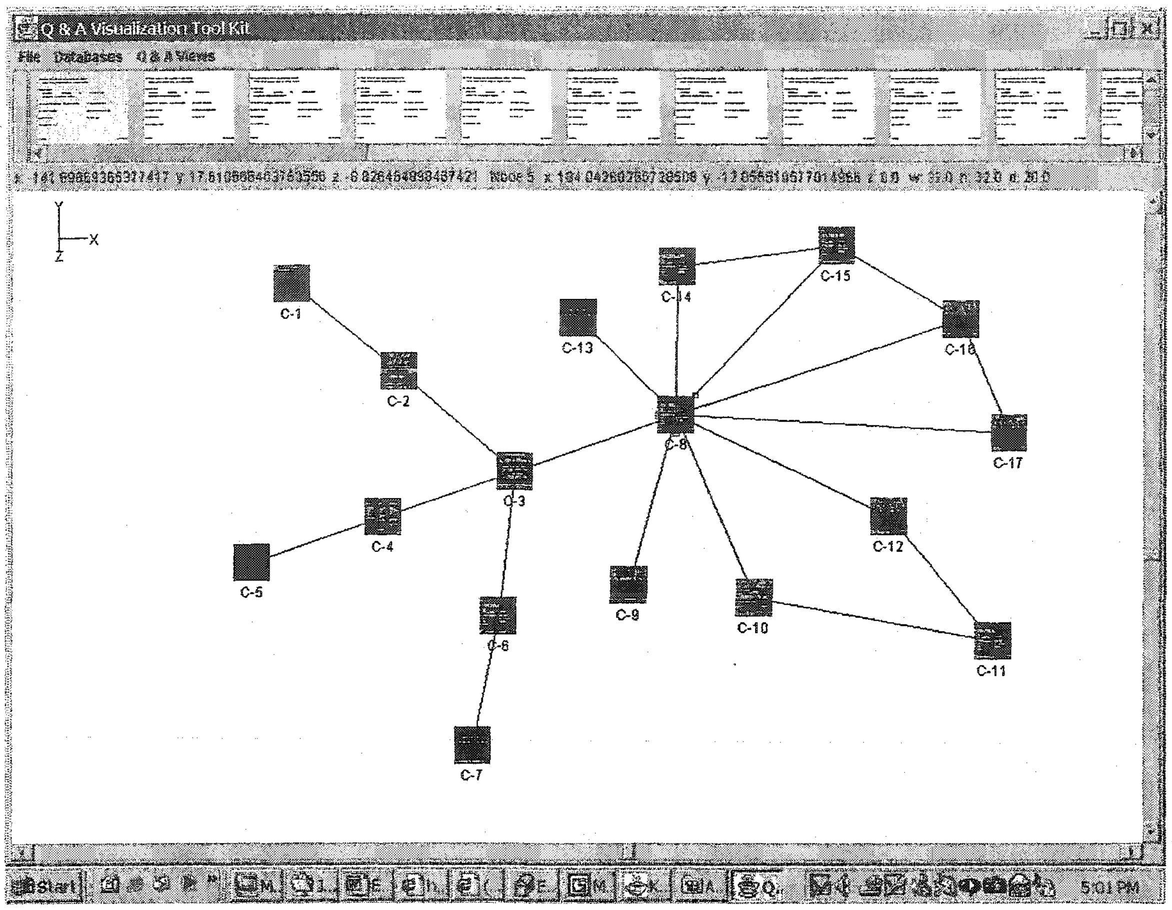


Figure 5.3. QandA Cluster Review User Interface. The Lower Panel Shows Clusters as Thumbnails. The Upper Panel Shows The Centroid and Snapshots of The Selected Cluster (C-8).

5.2.4 A Metric for Measuring Clustering Quality

LeNDI needed a metric for assessing the quality of the outcome of its clustering process, whether it is done using the incremental clustering algorithm or the top-down clustering algorithm. The metric should be able to measure the distance of a produced partition from a reference or authoritative partition, i.e., a partition that has been constructed and/or verified by the LeNDI analyst and is considered to be the truth. In other words, after producing a partition of a set of snapshot traces, the LeNDI analyst would fix any mistakes in clustering by moving misclustered snapshots to where they should belong, thus producing a correct or authoritative partition. Then, such a metric should gauge a meaningful distance between the derived and the authoritative partitions. For both clustering algorithms, this metric is important to measure how well the algorithms work for a given set of data. For the top-down algorithm, user feedback is used

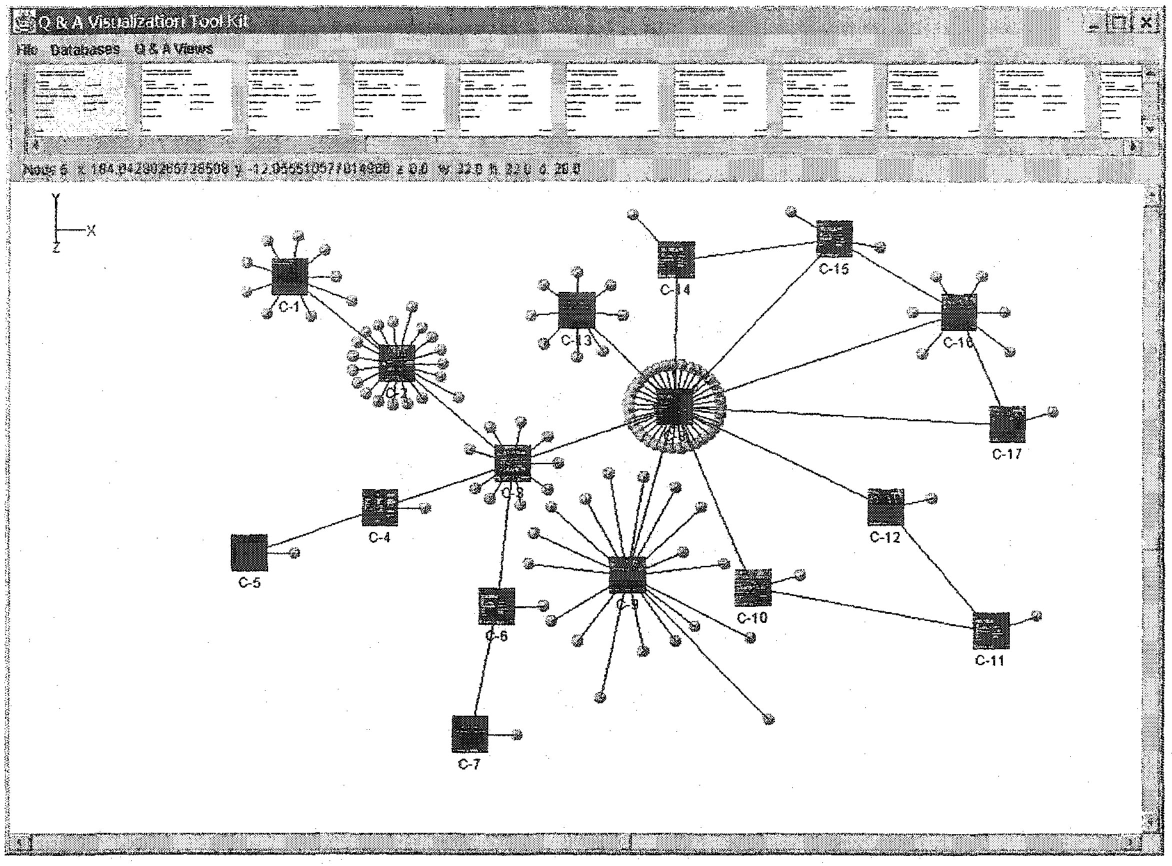


Figure 5.4. QandA Snapshots View User Interface. (The line connecting a snapshot to its cluster centroid shows the distance between them, and hence, their similarity).

for classifier induction by producing an enhanced decision tree as described in subsection 5.3.2. To obtain such a similarity measure, MoJo metric [TH99] was adopted and extended to MoJo Plus. MoJo uses a heuristic to approximately count the minimum number of operations that are required to transform one partition to another. MoJo allows only two operations, ***MOVE*s** and ***JOIN*s**. A MOVE consists of moving a single instance from one cluster to another, while a JOIN merges one cluster into another.

5.2.4.1 MoJo Plus Metric

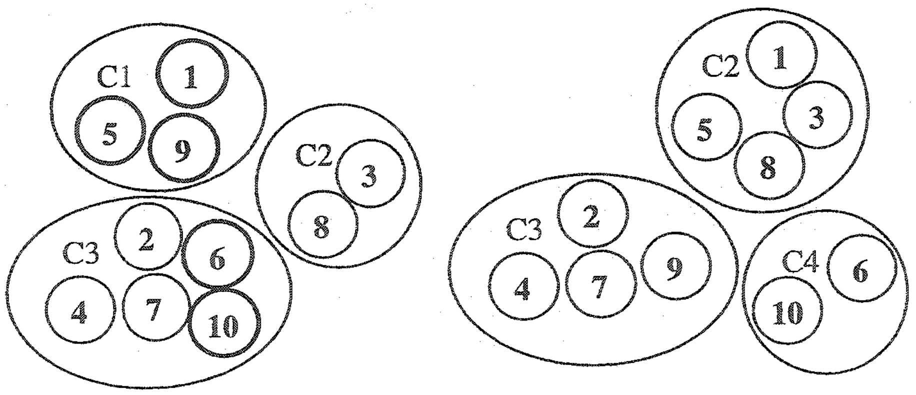
During our experiments, it was noted that MoJo metric does not adequately reveal the similarity of partitions that contain clear groupings of misclustered instances. In the context of snapshot clustering, it is frequently the case that a number of potentially pre-grouped snapshots have been included in one cluster and need to be moved together to another cluster. Selecting this easily identifiable group and determining where it should go is not much more effort than relocating a single misclustered instance. To include this fact in the metric, a "*M*ulti-*M*ove" or *MM* operation was added that considers moving a group of instances from one cluster to another as a single operation whose cost is equal to that of a JOIN or a simple single instance MOVE. This extension is named MoJo Plus [EISSM01]. The following example clarifies how the MoJo Plus metric is used.

5.2.4.2 A MoJo Plus Example

Consider a trace *t* with 10 snapshots, where *t* = {1,2,3,4,5,6,7,8,9,10}. Consider the two partitions shown for this trace in Figure 5.5. The left partition is derived while the one in the middle is an authoritative partition. In the far right, there are the MoJo Plus steps needed to transfer the derived partition to the reference one by changing the locations of the items in bold circles. If each step is given a weight, the distance between the two partitions can be calculated. If they all have a weight of one, then the distance between the two partitions is 3. The three MoJo Plus steps needed to fix the derived partition are a MOVE, a JOIN and a MM, respectively.

5.3 Screen Classifier Induction

Given a set of interaction traces, the LeNDI analyst usually would do multiple clustering iterations with different parameters until producing a near perfect partition, i.e., the best partition, in her/his judgement, that can be obtained using automated clustering. Next, the LeNDI analyst revises the produced partition to produce an authoritative reference partition for the given trace set. Revisions include joining redundant clusters with their originals and moving any misclustered snapshots to the right clusters. Using the revised reference partition, LeNDI induces a classifier that is able to classify new incoming snapshots at runtime to one of the available clusters in the revised partition.



MOVE	C1 (9) C3
JOIN	C1 C2
MM	C3 (6 10) C4

Figure 5.5. A Mojo Plus Example with a Derived Partition (left), a Reference Partition (middle) and The MoJo Plus Steps to Transform The First to The Second.

LeNDI implements two classifier induction methods. The first is a signature-based classifier. Its underlying idea is to infer a predicate for each legacy screen that is able to uniquely distinguish its snapshots. This is done by building a signature for every cluster that captures the commonality of its member snapshots. The second is an extended decision tree classifier that results from extending the decision tree produced by the top-down clustering algorithm to accommodate the feedback done by the user to fix the partition produced by clustering. The following is a detailed description of both classifiers. Section 5.5 provides and evaluation of both methods and a comparison with the benchmark decision tree algorithm, C4.5 [Qui93].

5.3.1 Classifier Induction Method 1: Screen Predicate (Cluster Signature) Calculation

This first classifier is a simple signature-based classifier. LeNDI infers a signature for every cluster, i.e., a logical combination of feature values that uniquely distinguishes the members of this cluster. A signature consists of an artificial feature vector and an artificial snapshot presentation space. The artificial feature vector captures all the feature values common in all the screen snapshots it represents and has an indifferent symbol '?' wherever a common value is lacking. The artificial presentation space captures the commonality of all the screen snapshots it represents and has one of a number of indifferent symbols wherever the snapshots differ.

To build the artificial feature vector, for every single-part feature and for every part of every multi-part feature, LeNDI checks if the same value exists in all snapshots. If yes, it adds this value for this feature or feature-part to the artificial feature vector. Otherwise, it adds '?'. For example, assume a cluster with three snapshots and with three features for each snapshot: one string feature, one discrete multi-part feature and one binary multi-part feature. Assume the following feature vectors for the three snapshots: ("Code 101", 10-8-9, 101010), ("Code 102", 14-8-1, 101011), ("Code 101", 10-8-1, 101011). The signature of this cluster would be (? , ?-8-?, 10101?).

Algorithm 5.3: Building a Signature Presentation Space for a Cluster
Input: A cluster *c* of snapshots
Output: An artificial signature presentation space *SigPres* for the snapshots of *c*
Steps:

```
1. $digits = 0$
2. $spaces = 0$
3. $sameChar$ = TRUE
4. For $i = 1$ to # snapshot rows
  5. For $j = 1$ to # snapshot columns
    6. For every snapshot $snap_k \in c$, $1 \leq k \leq |c|$
      7. If $snap_k[i][j]$ is SPACE then $spaces$++
      8. If $snap_k[i][j]$ is DIGIT then $digits$++
      9. If $k > 1$ then
        10. If ($snap_k[i][j]$ != $snap_{k-1}[i][j]$) then $sameChar$ = FALSE
    11. If ($sameChar$ == TRUE ) then $SigPres[i][j] = snap_1[i][j]$
    12. Else
        13. If ($digits == |c|$) then $SigPres[i][j]$ = '!'
        14. If $spaces > 0$ then $SigPres[i][j]$ = '~'
        15. Else then $SigPres[i][j]$ = '?'
```

Algorithm 5.3. Building a Signature Presentation Space for a Cluster.

To build the artificial presentation space, LeNDI follows Algorithm 5.3. The algorithm takes as input a cluster *c* of snapshots. It outputs an artificial presentation space that is formed by superimposing the snapshots of *c* and analyzing the content of each superimposed location. For every position in the artificial presentation space matrix *SigPres* [*i*][*j*], the algorithm checks the corresponding positions in all the snapshots of the given cluster and counts the number of spaces and digits and checks whether all these positions have the same character (steps 4 to 10). Then, it sets *SigPres* [*i*][*j*] according to the findings. If the same character (any character) exists in all the snapshots, *SigPres* [*i*][*j*] is set to this character (step 11). If there is always some digit in this position, but not the same digit, then *SigPres* [*i*][*j*] is set to '!' (step 13). Otherwise *SigPres* [*i*][*j*] is set to one of two indifferent characters, '~' or '?' (steps 14 and 15). The first is more general as it means there is sometimes a character in this position, while in other times there is a space. The second means that some character (non-space) always exists.

The signatures produced by Algorithm 5.3 are the identity predicates of the nodes of the state-transition model that distinguish the members of the corresponding clusters. At runtime when a new snapshot is received, its feature vector can be computed and matched against all the cluster signatures available. If the new feature vector matches a single artificial feature vector, then LeNDI recognizes the snapshot as an instance of the corresponding cluster. A match here means that wherever there is a value in the signature feature vector, the same value exists in the snapshot's feature vector. While, wherever there is an indifferent character ('?') in the signature feature vector, no comparison takes place. If the snapshot's feature vector matches no signature feature vector, LeNDI would get lost in the CUI. This may mean that the new snapshot is an instance of a never seen before screen that does not have a corresponding cluster or node on the state-transition model. The reengineered UI needs to be equipped with a method to deal with such situations. An example for such a method may be issuing one or a series of reset actions that returns the CUI and the reengineered UI to the starting point before starting executing the task in hand. If the snapshot's feature vector matches more than one signature, LeNDI matches the presentation space of the given snapshot against the artificial presentation spaces of all the matching signatures. If only one signature presentation space matches the given snapshot, LeNDI recognizes the snapshot as an instance of the corresponding cluster. In case of multiple-presentation space matches, LeNDI would not be able to decide on its own the proper classification of the new snapshot from among the matching signatures. Thus, the reengineered UI developed in the forward engineering phase of CelLEST need to be equipped with a method to resolve such ambiguity. For example, if one of the two or more possible states is expected according to a task plan, then it would be possible to disambiguate accordingly.

Signature matching is done using Algorithm 5.4. Steps 1 and 2 iterate over every location or cell on the given snapshot *snapPres* [*i*][*j*]. Steps 3 to 7 compare *snapPres* [*i*][*j*] with the corresponding cell in the given signature's presentation space *signPres* [*i*][*j*]. If all the comparisons fail for all locations, then the algorithm returns TRUE. If any of the tests succeeds even for one location on the snapshot, then matching failure or FALSE is reported. Step 3 tests if *snapPres* [*i*][*j*] and *signPres* [*i*][*j*] contain the exact same letter. If they do not, step 4 checks if *signPres* [*i*][*j*] contains the very generic indifference letter, '~'. If it does not, step 5 checks if *signPres* [*i*][*j*] contains the indifferent digit character '!' while *snapPres* [*i*][*j*] is not a digit and step 6 checks if *signPres* [*i*][*j*] contains the indifferent non-whitespace character '?' while *snapPres* [*i*][*j*] is a whitespace.

```
Algorithm 5.4: Matching a Snapshot against a Signature Presentation Space
Input: A snapshot presentation space snapPres and a signature presentation space
       signPres
Output: TRUE if the snapshot matches the signature and FALSE otherwise
Steps:
1. For i = 1 to # rows of snapPres
   2. For j = 1 to # columns of snapPres
      3. If (snapPres [i][j] != signPres [i][j]) then
         4. If (signPres != '~') then
            5. If (signPres == '!') && (snapPres is NOT a digit) then return FLASE
            6. If (signPres == '?') && (snapPres is a whitespace) then return FLASE
7. Return TRUE
```

Algorithm 5.4. Matching a Snapshot against a Signature Presentation Space.

5.3.2 Classifier Induction Method 2: Decision Tree Extension via Supervised Learning

Once a preliminary clustering of the given snapshots is derived through the top-down clustering algorithm, the LeNDI analyst can examine the partition through QandA and move misclustered snapshots to their correct clusters and join redundant clusters with their originals. Then, s/he can request an extended decision tree using the decision tree extension algorithm[6] [EISSM01]. This algorithm leverages the decision tree produced by clustering, using the set of JOINs, MOVEs and MMs discovered by MoJo Plus. Algorithm 5.5 shows the pseudocode of this algorithm.

A MoJo Plus JOIN of cluster c_a to cluster c_b means changing the leaf node c_a to c_b or vise versa, or if c_a and c_b share the same decision node, replacing the decision node by the leaf node c_a or c_b. A MoJo Plus MOVE or MM of instances from c_{from} to c_{to} requires discovering which features distinguish the moved instances from the other instances in c_{from} and/or which features are shared by the moved instances and all the instances in c_{to}. The heuristic involved in this decision is to use the strongest and most generalizable distinguishing characteristics detected. If instances are being moved from a larger cluster to a smaller cluster, the tree extension algorithm first looks for features shared by the instances in the larger set, and not the instances being moved or in the destination set. If the instances are going from a smaller cluster to a larger one, the tree-revision algorithm

[6] This algorithm was developed primarily by Paul Iglinski, with help from other CelLEST team members.

first looks for features shared by the moved instances and the destination set, but not by the ones in the origin set. If the first of these feature quests fails, the alternative is tried next. If that second quest fails, the algorithm recursively tries again, this time ignoring the features in the destination set. If this still fails, the moved instances are split into groups according to the best-split test for minimizing the maximum internal cluster incoherence, and then each resulting group is checked recursively for distinguishing features. If the algorithm recurses down to a single instance, and no distinguishing feature can be found, the algorithm simply reports the failure and proceeds. This situation is, in fact, seldom encountered in all the legacy system traces that we have tested. The successful identification of distinguishing features guarantees the correct classification of the training set instances, i.e., the snapshots of the input set of traces. However, the problem of "getting lost" in the legacy CUI due to failure in classifying a new snapshot, as mentioned in subsection 5.3.1, can still occur and would need to prepare the reengineered UI to deal with it. Once a set of distinguishing features has been found, the algorithm currently selects one at random and uses it to create a new decision node to distinguish the instances from their initial classification. We experimented with various heuristics for selecting among a set of distinguishing features, and we evaluated their effectiveness with 10-fold cross validation. None proved more reliable than random selection.

Algorithm 5.5 takes as input a decision tree created by Algorithm 5.2 and a set of MoJo Plus moves that reflect the feedback of the LeNDI analyst to fix the derived partition, and its corresponding decision tree. Algorithm 5.5 starts by creating two Nodes, *nodeA* and *nodeB*. “Node” is a data structure that represents a decision node or a leaf node in the decision tree produced by Algorithm 5.2. If a leaf node is created, then a cluster will be associated with it. Step 2 creates 3 empty clusters to use during the algorithm. Step 3 iterates over every move in the given set of MoJo Plus moves. Step 4 checks if a given *move* is a JOIN, and if so, steps 5 to 9 are executed. If the move is MOVE or MM, then steps 11 to 15 are executed. Steps 5 and 6 retrieve the leaf nodes of the clusters to be joined and store them in *nodeA* and *nodeB*. If both nodes share the same parent, then they are both removed and their joint parent is set as leaf node whose cluster is that of *nodeA*. If *nodeA* and *nodeB* do not share the same parent, then the cluster of *nodeB* is set to that of *nodeA*. In other words, both branches are made to result in the same decision. In case of a MOVE or MM, step 11 retrieves the leaf node of the source cluster that will have some of its snapshots moved to a different cluster. Steps 12 to 14 retrieve the source and destination clusters, c_{from} and c_{to}, and the snapshots to be moved c_{moved}. Step 15 calls SplitNodesOnFeatures function and passes the data retrieved in steps 11 to 14 as parameters.

The SplitNodesOnFeatures function takes as parameters three clusters, c_{from}, c_{to} and c_{moved} and the leaf node *nodeA* of the source cluster. It splits the given node such that the snapshots of c_{moved} are separated from the rest of the snapshots in c_{from}, which are not in c_{moved}. This means that *nodeA* changes to a decision node and two leaf nodes are created. The function tries to discover which features distinguish c_{moved} instances from the other instances in c_{from} and/or which features are shared by c_{moved} instances and all the instances in c_{to}.

Algorithm 5.5: Classifier Induction via Decision Tree Extension
Input: A decision tree *Tree* produced by the top-down clustering Algorithm 5.2 and a list of MoJo Plus moves *Moves* to fix *Tree*
Output: The decision tree *Tree* after applying *Moves* to it

```
1. Create new Nodes $nodeA$, $nodeB$
2. Create new clusters $c_{from}$, $c_{to}$, $c_{moved}$
3. For each $move \in Moves$
    4. If $move$ = JOIN
        5. $nodeA$ = Leaf Node of $Tree$ Corresponding to $move$.getFirstCluster()
        6. $nodeB$ = Leaf Node of $Tree$ Corresponding to $move$.getSecondCluster()
        7. If ($nodeA$.getParentNode() == $nodeB$.getParentNode())
            8. Set Parent of $NodeA$ as a Leaf Node whose cluster = $nodeA$.getCluster()
        9. Else Set cluster of $nodeB$ = $nodeA$.getCluster()

    10. Else
        11. $nodeA$ = Leaf Node of $Tree$ Corresponding to $move$.getFromCluster()
        12. $c_{from}$  = $move$.getFromCluster()
        13. $c_{to}$    = $move$.getToCluster()
        14. $c_{moved}$ = $move$.movedInstances()
        15. SplitNodesOnFeatures ($c_{from}$, $c_{to}$, $c_{moved}$, $nodeA$)

SplitNodesOnFeatures (Cluster $c_{from}$, $c_{to}$, $c_{moved}$, Node $nodeA$)
1. If ($|c_{from}| > |c_{to}|$) then
    2. Split $c_{from}$ on shared features of $c_{from} - c_{moved}$ not shared by $c_{to}+c_{moved}$
    3. If split is successful then return
    4. Split $c_{from}$ on shared features of $c_{to}+c_{moved}$ not shared by $c_{from} - c_{moved}$
    5. If split is successful then return

6. Else
    7. Split $c_{from}$ on shared features of $c_{to}+c_{moved}$ not shared by $c_{from} - c_{moved}$
    8. If split is successful then return
    9. Split $c_{from}$ on shared features of $c_{from} - c_{moved}$ not shared by $c_{to}+c_{moved}$
    10. If split is successful then return

11. If ($c_{to}$ != NULL) then
    12. SplitNodesOnFeatures ($c_{from}$, NULL, $c_{moved}$, $nodeA$)
    13. Return

14. If ($|c_{moved}|>1$) then
    15. Split $c_{moved}$ according to its best split (Algorithm 5.2) and store the result in $c_1$ and
        $c_2$
    16. SplitNodesOnFeatures ($c_{from}$, $c_{to}$, $c_1$, $c_1$.getLeafNode())
    17. SplitNodesOnFeatures ($c_{from}$, $c_{to}$, $c_2$, $c_2$.getLeafNode())

18. Else Return Message "Split Failure"
```

Algorithm 5.5. Classifier Induction via Decision Tree Extension

Step 1 checks if the size of c_{from} is bigger than that of c_{to}. If so, then step 2 tries to split c_{from} into two clusters. The first cluster is c_{moved} and the second is c_{from} - c_{moved}. The split decision is based on a feature whose value is shared by all the instances of c_{from} - c_{moved} and by none of the instances of c_{to} or c_{moved}. If such a feature is found, then step 3 returns to the main algorithm. If the split fails, then step 4 tries a new split based on any feature whose value is shared by all the instances of c_{to} and c_{moved} and by none of the instances of c_{from} - c_{moved}. If such a feature is found, then step 5 returns to the main algorithm. If the split fails, then the algorithm moves to try an easier split at step 11. Steps 7 to 10 are executed if the size c_{from} is less than or equal that of c_{to}. They perform the same tests as steps 2 to 5 but reversed, i.e., moving from the stronger test to the weaker one[7]. The test of step 11 is performed only if no split was successful in steps 1 to 10. It tests if the parameter c_{to} is not an empty cluster. If it is not, it calls the function SplitNodesOnFeatures with NULL passed to the parameter c_{to}. This means that the tests and trials to split of steps 1 to 10 will be repeated but with ignoring the instances of c_{to} as if every reference to c_{to} in steps 1 to 10 is blank, making splitting efforts easier. This is because a smaller number of snapshots will be involved in trying to split c_{from}. If in the second call of SplitNodesOnFeatures, where NULL was passed to c_{to}, steps 1 to 10 fail to split c_{from} or if the function was called the first time with NULL passed to c_{to}, steps 14 to 18 are executed. Step 14 checks if there is more than one instance in c_{from}. Steps 15 to 17 split c_{from} using the best-split test of Algorithm 5.2 into c_1 and c_2. Then, they call the function SplitNodesOnFeatures twice, one time for each of c_1 and c_2. This process repeats recursively. If the algorithm recurses down until SplitNodesOnFeatures is called with only one instance in c_{from} and it was impossible to split it apart from the rest of c_{from}, step 18 reports failure and proceeds.

5.4 Transition Modeling

Transition modeling aims to infer accurate behavioral models that describe the permissible user behaviors on every state of the state-transition model, i.e., the actions available to the legacy system user for navigating from one CUI screen to another. These models are the edges connecting the nodes of the state-transition model, as shown in

[7] By stronger, we mean that involves a bigger number of instances.

Figure 5.1. Modeling these edges does not only complete the state-transition model, but also has other benefits. They can be used as features for screen snapshot identification. They are necessary for planning a navigation sequence to accomplish some task. Also, they provide some of the information necessary for user task modeling, and thus save some effort in the modeling process. A formal definition of a transition was given in Definition 5.1.

Most legacy interfaces adopt a mix of function key, menu driven, command-driven, and form-filling interaction styles. In the function key interaction style, the interface implements a well-structured dialog with the user. At each point of this dialog, the user presses one of a small set of function keys to select one of the corresponding alternative options. A similar kind of interaction can be implemented in a menu driven interface. Such an interface presents the user with a list of items, each of which can be selected by moving the cursor to its location and pressing a control key. In the command-driven interaction style the user issues textual commands to the system. A command language is specified in terms of the vocabulary of possible command names and the syntax of these commands in terms of the arguments they require and the options they allow. The command-driven interaction style enables more dynamic system user interaction, since the transitions of the system from one state to another are caused by possibly complex, multi-parametric commands instead of simple function key presses. Finally, in the form-filling interaction style, the interface presents the user with forms that require the entry of specific types of information at particular locations on the screen. The completion of the form is signaled to the system with the press of a control key or the typing of a command at a particular command line. The current version LeNDI has focused on systems adopting a combination of function key and command-driven interaction styles, which is a frequently occurring combination.

```
Transition    := <Start Screen><Action><End Screen>
Action        := <Action Item>+
Action Item   := <Location><Data Item>|
                 <Location><Data Item><Control Key>
Location      := <x-y coord> | <Range> | φ
x-y coord     := [1,80],[1,24]
Range         := x-y coord, x-y coord
Data Item     := <Keyword>*<Argument>*<Option>*
Keyword       := String ∈ Set of possible keywords
Argument      := String
Option        := String ∈ Set of possible options
Control Key   := PF1 | PF2 | ...... | Enter |
Start Screen  := <Screen Id>
End Screen    := <Screen Id>
Screen Id     := Integer
```

Figure 5.6. A Grammar for Describing Transitions in Legacy Systems CUIs (A '*' is zero or more occurrences, a '+' is one or more and φ is Null).

5.4.1 A General Model for Transitions

LeNDI possesses a general transition model, described by the BNF (Backus Naur Form) grammar shown in Figure 5.6. It was developed to describe the various styles of interaction mentioned earlier. According to this model, each *transition* from a *start screen* to an *end screen* is caused by an *action,* which may consist of one or more *action items*. An action item may involve a data-entry activity by entering a *data item* on a particular *location* of the screen, which may be static or dynamic, i.e., varying within a *range*. A range is a rectangular area defined by the x-y coordinates of its upper left and bottom right corners inside which the data item starts, i.e., its first character exists. An action item may conclude with the press of a *control key*. A data item can have *keywords*, *arguments* and/or *options*.

To perform transition modeling for function key and command-driven interaction styles, LeNDI groups the snapshots of each cluster according to the destination of the user action performed on them. LeNDI assumes that there is a single action leading from a start screen to an end screen, although the action may have different forms. Therefore all the transitions in one group have instances of the same action. Next, LeNDI tries to infer the command form(s) and/or the function key(s) that defines this action, assuming that it conforms to the general model described by the transition-model grammar.

LeNDI starts analyzing each group of action instances, one word at time, starting with the first word in all instances. LeNDI uses a set of rules for command language design [Sch99] to discover any relations between the most frequent 4 words that appeared as an action's first word and whether any of them is an optional or mandatory command keyword or argument. According to these rules, LeNDI assumes that if there are different versions of the same command name they will most likely be prefixes of a "canonical" command name or sub-strings of this name with the vowels removed. In order for a particular string to be identified as the "canonical" command keyword, its different variants have to appear frequently enough, i.e., at least 33% of the times that the action occurred. This analysis of command keywords applies to command arguments too. If a number appears at least 33% of the time, LeNDI concludes that one form of the command keyword or argument is a numerical. If no keyword appears sufficiently often, LeNDI assumes that the command name is implicit, and that the user has to only enter its arguments. It assumes that an argument is optional if it does not appear in some of the action instances, otherwise it assumes that it is mandatory. The same analysis is applied to the second word, and so on. LeNDI assumes that the command keyword, if any, can be at any position, and is not necessarily the first word. LeNDI collapses the collected hypotheses in a compact form. LeNDI analyzes the recorded locations of all the instances of an action to infer any information about where it takes place on the legacy screen. Using simple comparison of the x and y coordinates of these instances, LeNDI defines the location or range within which the action takes places.

5.4.2 Transition Modeling Examples

Two transition-modeling examples are shown in Figures 5.7 and 5.8. The example of Figure 5.7 represents modeling the transition from the results of *browse* command screen to the same screen in a command-driven library system (LOCIS) [LOCIS], i.e., self-transition. 30 instances of the action that causes this transition are shown in Figure 5.7(a). LeDNI starts the analysis by analyzing the first word in all instances. It discovers that *b*, *brws* and *browse* are all derived from the same canonical form, which is *browse*, either by suffix removal (*b*) or by removal of vowels (*brws*), respectively. The different forms of *browse* command are repeated 22 times in the 30 instances. The conclusion is that there is a form of this command that requires one of these three command variants to exist as the first word of the action needed for the transition. Since the remaining 8 instances do not have any data items, just a control key (Enter), LeNDI concludes that a second form of the action consists only of *Enter* key. Similar analysis for the second and third words in the 22 instances with a keyword, concludes that no word or a group of related words appears in any of these positions in at least 33% of the instances. Also, some instances lack words in the second and third or third locations. Thus, according to the given instances, the command can have up to two optional arguments. By analyzing the locations of these instances, LeNDI concludes that the action instances with a keyword and arguments occur within rows 21 to 23 and always on column 11, while those with *Enter* key occur always at row 23 and column 11. The inferred model is shown in Figure 5.7(b). * is a mandatory argument and [*] is an optional argument

Figure 5.8 shows the second example. It is modeling the transition from a *browse* command results screen to a *retrieve* command results screen in LOCIS, by issuing a *retrieve* command. Seven instances of this transition were recorded. LeNDI examines the first word, which is *R* for all the examples and concludes that it is a compulsory keyword for this action. Then it examines the second word. By applying the rules mentioned above, no relation can be discovered between the words in the second position and none of them appears more than 33% of the time. Thus, the second word is considered a mandatory argument for the command because all the instances have a second word. Doing the same analysis for the third word concludes that it is an optional argument. Finally, by comparing the locations of all action instances, LeNDI infers that it always takes place at a fixed location. The inferred model is shown in Figure 5.8(b).

LeNDI has a transition viewer tool, which allows reviewing all the action available on each screen (node) of the state-transition model, their different forms and their destination screens. For each from of an action, one can review the instances that were used to infer this form.

	Row	Col.	First Word	Second Word	Third Word	Control Key
1	23	11	b	Ali		*Enter*
2	23	11	b	b6\		*Enter*
3	23	11	B	b6		*Enter*
4	23	11				*Enter*
5	23	11				*Enter*
6	23	11				*Enter*
7	23	11				*Enter*
8	23	11				*Enter*
9	23	11	b	Elections		*Enter*
10	22	11	brws	rep	smith	*Enter*
11	23	11				*Enter*
12	23	11				*Enter*
13	23	11				*Enter*
14	21	11	b	Linda	Smith	*Enter*
15	23	11	b	elections--		*Enter*
16	23	11	b	astronomy—bibliography		*Enter*
17	23	11	b	term/iran		*Enter*
18	23	11	brws	text/b6		*Enter*
19	22	11	Browse	c97/egypt		*Enter*
20	23	11	b	subj=b6		*Enter*
21	23	11	b	b6		*Enter*
22	23	11	b	subj=b11		*Enter*
23	23	11	Browse			*Enter*
24	23	11	Browse	egg	Research	*Enter*
25	22	11	b	r6		*Enter*
26	22	11	b	r9		*Enter*
27	22	11	b			*Enter*
28	22	11	b			*Enter*
29	22	11	b	R	2	*Enter*
30	23	11	b	r5		*Enter*

(a) The action instances

```
b_brws_browse [*][*]     @Enter  @ [21,23],11
[]                       @Enter  @ 23,1
```

(b) The inferred command model

Figure 5.7. An Example (1) of Transition Modeling in a Command-driven System.

	Row	Col.	First Word	Second Word	Third Word	Control Key
1	24	11	R	Farm	Loans	*Enter*
2	24	11	R	xxx		*Enter*
3	24	11	R	term=tax	Deductions	*Enter*
4	24	11	R	Tax	Deductions	*Enter*
5	24	11	R	b6		*Enter*
6	24	11	R	s1		*Enter*
7	24	11	R	elections		*Enter*

(a) The action instances

```
R * [*] @Enter @24,11
```

(b) The inferred command model

Figure 5.8. An Example (2) of Transition Modeling in a Command-driven System.

5.5 Evaluation

This section reports some of the experiments done to evaluate the behavior modeling process presented in this chapter and to explore its strengths, limitations and potential future enhancements. It reports experiments done with publicly accessible systems, since some other LeNDI evaluation experiments were done using private data obtained from Celcorp, the industrial sponsor of CelLEST project. Three different experiments are presented in this section to serve different evaluation purposes. The first and second experiments are done for the purpose of comparing and evaluating the single-path incremental and the top-down clustering algorithms and the associated classifiers against one another and against C4.5 [Qui93]. The results of both experiments are used to explore the strengths and weaknesses of both algorithms. The third is a complete case study of behavior modeling, performed on a long trace recorded during interaction with an information system for a university library research network. In this experiment, the CUI of a big selected part of the system was modeled. This experiment demonstrates the applicability and efficiency of the method.

5.5.1 Experiment 5.1 - LOCIS System

In this section, we report the results of an experiment with an IBM 3270 trace of interaction with LOCIS through its public 3270 connection. It was recorded while a user was browsing the library catalog, retrieving sets of catalog entries, displaying them, and running into some system errors. This trace is 406 snapshots long. Manually, an analyst built an authoritative partition for this trace, which had 17 distinct clusters. The number of instances of each cluster of the authoritative partition is shown in the "Cardinality" row in Table 5.1. Note that the data set is unbalanced: some screens had only 1 or 2 snapshots in the trace, while others had up to 157. Figure 5.1 depicts a segment of the LOCIS trace and a part of the derived model.

5.5.1.1 Modeling Using Single-Path Incremental Clustering and Signature-based Classification

Typically, the single-path incremental clustering algorithm requires several rounds of configuration, clustering and result review until reaching satisfactory results. The efficiency and accuracy of the resulting partition depends on the intuition and experience of the analyst. This experiment was performed by a user who had no particular familiarity with LOCIS but was familiar with the overall CelLEST process and was given a tutorial on using LeNDI. Out of the discrete feature suite of LeNDI (Table 4.3) the user created only one recognizer with the feature set of Table 5.2.

Cluster Id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Cardinality	11	14	157	16	105	6	15	7	3	2	13	1	1	34	5	5	11

Table 5.1. The Reference Partition Cardinality of The Data Set of Experiment 5.1.

Feature	Description	Weight	Ignore if Empty
1-3	The text in the middle of the first non-blank row	30	N
5-1	The cursor's label	10	N
6-2	All characters binary horizontal profile	20	N
6-5	Special characters binary profile	10	Y
7-1	Layout classification	5	Y
7-2	Layout specifications	25	N

Table 5.2. The Features Used for Setting up The Single-path Incremental Clustering Algorithm for LOCIS Experiment 5.1.

A threshold of 40% was used and the cluster centroid was defined to be its representative. It took eight recognition/review/reconfiguration rounds to reach the setup shown in Table 5.2, which the user thought was satisfactory. The column "weight" gives the relative weight of each feature compared to other features. The column "Ignore if empty" indicates whether to ignore a feature if missing on some snapshot, or not.

The partition produced by the final configuration consisted of 23 different clusters. It included 17 misclustered snapshots or outliers (4.2%) and 6 redundant clusters. An outlier is a false positive error that assigns snapshots with potentially different behavior to the same screen cluster, causing false connections between the state-transition graph nodes. Redundant clusters are considered false negative errors which are duplications in the state-transition graph, resulting from the snapshots of the same screen being split into two or more clusters. This partition was reviewed by the user and 12 corrective operations, i.e., JOINs, MOVEs and MMs operations, were necessary to fix the errors identified. After, moving the misclustered instances to their clusters, a signature was calculated for every cluster using the signature calculation algorithm. When the generated signature was used to recognize the trace snapshots one snapshot was misclustered (0.25%); it matched more than one signature and was assigned to one of them randomly. Such problems of signatures with overlapping applicability result from the diversity of the instances in some clusters, which results in little commonality among their snapshots and hence, "weak signatures". Ideas to overcome this are given in Section 5.6.

A better measure of the algorithm's performance on unseen data was obtained with repeated 10-fold cross validation. This means that the data was divided to 10 equal parts and the experiment was repeated 10 times, with one part used as a test set in each round and the remaining 90% is the training set. 10-fold cross validation using single-path incremental clustering yielded an error rate of (8%) (see Table 5.3) on LOCIS data set. The second and third columns in Table 5.3 show the distance between the partition produced by the clustering algorithm and the authoritative partition built manually in MoJo and MoJo Plus moves, respectively. The fourth column is the percentage of snapshots used to induce the signature classifier that were misclassified. The fifth column is the average test error of the repeated 10-fold cross validation.

5.5.1.2 Modeling Using Top-Down Clustering and Decision Tree Classification

For the same LOCIS trace, the 39 binary features of LeNDI were extracted for every snapshot of the trace. Then, the top-down clustering algorithm was applied to the data with an input parameter of 17 clusters – the expected number of clusters was already known from the authoritative partition built by the user in the previous experiment. The cardinalities of the produced clusters are shown in Table 5.4 and the corresponding cluster Ids in the authoritative partition. Three from the authoritative partition were missing in the derived partition. 14 (3.4%) snapshots were clustered into 3 redundant clusters. On the other hand, 44 (10.8%) snapshots were misclustered. Ignoring the 3 unnecessary splits, we can say that 89.2% of the instances were “correctly” clustered. The partition was again reviewed and revised by the user. Using QandA, the user corrected the preliminary clustering of the LOCIS trace. Then, MoJo Plus module inferred the operations necessary to obtain the desired authoritative partition, which are shown in Figure 5.9. The decision tree extension algorithm was applied, and a new tree containing 46 nodes and having a maximum depth of 12 was produced. When this decision tree was tested on the 406 snapshots, all were correctly classified. 10-fold cross validation on LOCIS yielded an error rate of 3.4% on the data in the test sets (see Table 5.3).

Clustering Method	MoJo Moves	# MoJo Plus Moves	Training Error	Test Error
Single-path Incremental Clustering with Signature-based Classifier	23	12	0.25%	8.0%
Top-down Clustering with Decision Tree Classifier	47	20	0.00%	3.4%
C4.5 (Supervised Learning)	NA	NA	1.20%	2.4%

Table 5.3. The Results of Experiment 5.1.

Cluster Id	0	1	2	3	4	5	**6**	7	8	9	10	11	12	13	**14**	**15**	16	Clusters missing after top-down clustering		
Corresponding Authoritative Cluster Id	8	1	14	3	9	11	-	16	4	10	6	17	7	2	-	-	5	12	13	15
Cardinality	7	11	34	157	3	13	**1**	5	16	2	6	11	15	14	**5**	**1**	105			

Table 5.4 The Results of Top-down Clustering of The LOCIS Trace of Experiment 5.1. Bold Clusters Are Redundant and Need to Join Others.

```
JOIN   C8 C6                        // Merge cluster C6 into C8
JOIN   C16 C14                      // Empty clusters are available as "new" clusters
JOIN   C2 C15                       // C15 is now empty. It is reused in the next step
MOVE  C0  (122)  C15                // Take snapshot 122 from C0 to C15
MM     C0  (306 304)  C11           // A Multi-move from C0 to C11
MOVE  C1  (309 308 307 305 303 302 301 300 299)  C11
MM     C1  (405 404)  C9
MOVE  C2  (318)  C13
MM     C2  (134 133)  C16
MM     C2  (317 316 315 278)  C14
MM     C3  (401 291 195 186 132 47)  C12
MOVE  C3  (400)  C2
MOVE  C3  (338)  C14
MOVE  C8  (101)  C5
MOVE  C9  (118)  C2
MM     C10  (379 375 371 279 5)  C13
MM     C10  (388 218 110 102)  C7
MOVE  C11  (399)  C2
MM     C16  (382 57)  C13
MOVE  C16  (406)  C6
```

Figure 5.9. The MoJoPlus Operations Needed to Fix the Clustering of The LOCIS Trace Using Top-down Clustering, in Experiment 5.1.

5.5.1.3 Comparative Evaluation

To evaluate the results obtained from the experiments above, we used C4.5 [Qui93], a standard decision-tree learning algorithm. C4.5 is a classifier-induction algorithm that takes labeled examples as input. Hence, we used it to evaluate the final outcome of the experiment in the form of a signature-based or decision tree classifier. Several versions of C4.5 were tried, using the 39 binary features of LeNDI. We tried pruned decision tree, unpruned decision tree and rule-based versions of C4.5. The best results were obtained using the pruned version of C4.5 and are reported in Table 5.3.

5.5.1.4 Transition Modeling

LOCIS is a command-driven system. Some of the command models inferred are shown in Figure 5.1(b). They are inferred from analyzing the entire trace not only the part shown in Figure 5.1(a). An example action model inferred is "*d_disp_display item* *". The command "display item" causes the transition from screen 6 to 7. In the inferred model, LeNDI discovered three variants for the first keyword and that the second keyword is *item*. Also, there is also a mandatory argument that has to be passed to the command.

5.5.2 Experiment 5.2 - HOLLIS System

This experiment is similar to Experiment 5.1, but it is performed on three interaction traces recorded while using Harvard Online Library Information System (HOLLIS) [HOLLIS] through its 3270 public connection. HOLLIS is a command-driven catalog of the millions of items at Harvard University Libraries, e.g., books, journals, manuscripts, government documents, visual materials, etc. Together, the three traces had 542 snapshots, which were instances of 29 distinct legacy system screens. They captured snapshots of the main user interfaces of the three subsystems of HOLLIS: Harvard Union Catalogue (HU), Reserved Material (RV) and Library Guide (LG). An authoritative partition was built for the input traces. Table 5.5 shows the cardinality of all the clusters in this partition. The results of this experiment are shown in Table 5.6. First, the single-path incremental clustering was used with signature-based classification. The final setup used this algorithm included one recognizer that utilizes only Feature1-3 with weight 100%. A threshold of 70% was used. In this experiment 491 (90.6%) snapshots were correctly clustered. 48 (8.9%) were correctly clustered, but in redundant clusters. The number of redundant clusters was 21. Only 3 (0.6%) snapshots were misclustered, i.e., put in clusters of snapshots of other screens. Training error was 0.6% and testing error using 10-fold cross validation was 1.7%. Second, the top-down clustering was applied with an input parameter of 29 clusters, followed by decision tree classifier induction. Training error was 1.0% and testing error was 4.3%.

Cluster Id	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29
Cardinality	16	7	56	57	8	12	107	10	11	82	1	4	2	4	6	28	2	6	10	9	14	1	6	3	1	14	60	3	2

Table 5.5. The Authoritative Partition of the Data Set Used in Experiment 5.2.

Clustering Method	MoJo Moves	# MoJo Plus Moves	Training Error	Test Error
Single-path Incremental Clustering with Signature-based Classifier	147	92	0.6%	1.7%
Top-down Clustering with Decision Tree Classifier	162	50	0.0%	5.4%
C4.5 (Supervised Learning)	NA	NA	1.0%	4.3%

Table 5.6. The Results of Experiment 5.2.

5.5.3 Comments on Experiments 5.1 and 5.2

This subsection comments on the results of Experiments 5.1 and 5.2 and draws some observations on the efficiency of the clustering and classifier induction methods used in LeNDI.

1. As can be seen from Tables 5.3 and 5.6, in both experiments, the training error of the decision tree classifier was 0, while that of the signature-based classifier and of C4.5 was not. For the signature based-classifier, this is due to imprecise signatures. A suggestion to overcome this problem is given in section 5.6. As for C4.5, having training error higher than both of LeNDI's two classifiers can be attributed to the unbalanced data set, i.e., the lack of sufficient examples in some clusters, as shown in Table 5.1. It shows that 2 clusters in LOCIS experiment have only one snapshot each, 6 clusters have 5 or less snapshots and 8 clusters have 10 or less snapshots. For the HOLLIS experiment, shown in Table 5.5, 3 clusters had only 1 instance, 10 clusters had 5 or less instances and 18 clusters had 10 or less instances.
2. The fact that all classifiers have larger test error than training error is not surprising; they tend to overfit the training data and do not generalize well enough. Again,

sufficient examples that cover the range of variability of the snapshots of each screen are necessary for producing high-quality classifiers. And as can be seen from Tables 5.1 and 5.4, clusters 12, 13 and 15 in the LOCIS experiment, which did not have equivalent clusters in the partition produced by the top-down algorithm, all had few instances of them in the input trace. They had 1, 1 and 5 instances respectively. This suggests that a sufficiently large number of snapshots need to be recorded from interactive online sessions with the legacy application. What qualifies as "sufficient" number depends on the number of CUI screens to be modeled, the types and dynamics of these screens (see section 4.1) and the productiveness of the extracted features for the system in hand. Similarly, the generality of the action models produced depends on the availability of sufficient instances and on action complexity.

3. Redundant nodes on the state-transition graph occur when the snapshots belonging to one state are split into more than one cluster. This is not a severe problem since it would not cause the runtime process to get lost, i.e., to misinterpret a snapshot as belonging to a wrong state. But false positive errors, resulting from misclustering snapshots, and possibly missing nodes for some of the interface screens, is a more serious error. It results in wrong assumptions about the screens (states) behavior. When a new UI or an external application uses the state transition model to execute a navigation sequence in the legacy CUI, such errors can cause incorrect predictions of the result of actions and result in the UI or external application "getting lost" in the legacy CUI.
4. The top-down clustering method with a decision tree classifier performed considerably better than the single-path incremental clustering method with a signature-based classifier in LOCIS experiment, but considerably worse in HOLLIS. This implies that the two methods can be complementary and therefore LeNDI's performance is improved by employing both of them. Top-down clustering is suitable when a reasonable estimate of the number of screen-states sought can be obtained. While, the incremental clustering is iterative and requires several cycles of user setup, clustering and result review. This requires good grasp of the system in hand and is useful if the analyst will invest time in exploring and learning the legacy CUI as s/he is modeling it.

5.5.4 Experiment 5.3 - MIRLYN System

This section reports the results of a case study to reverse engineer the legacy CUI of an IBM 3270 legacy system and build its state-transition model. The purpose of this modeling case study is to validate the practicality and efficiency of the legacy CUI behavior modeling process described in this chapter, rather than evaluating or comparing individual algorithms. Four traces were used in this experiment with 1924 snapshots in total. 64.3% of this data was used for training and 35.7% for testing

In this experiment, the author modeled part of the legacy CUI of MIchigan Research LibrarY Newtork (MIRLYN) [MIRLYN]. This information system is a catalog for the different resources available in the libraries of University of Michigan and the remote libraries of other institutes that are connected together under MIRLYN. The number of states of the entire system is huge due to the many local and remote subsystems and indices connected. Therefore, this experiment focused on building a behavior model for part of the MIRLYN CUI. This part covers the main catalog and the basic functions of a few subsystems and remote catalogs. The system was accessed through its publicly accessible IBM 3270 connection.

A user interacted with the system via LeNDI for a full day querying the different available local and remote catalogs about library items, searching for course reserve materials and doing other related information retrieval tasks with the system. The user was directed to which subsystems and catalogs to use and was asked to cover them thoroughly several times to ensure having enough examples of each screen. The user activity was recorded in one long trace of 1237 snapshots. The author modeled the behavior of the legacy system CUI by performing clustering, classifier induction and then transition modeling as described in details in the next subsections.

5.5.4.1 Snapshot Clustering

Since the author was not familiar with the legacy system, he decided to use the single-path incremental clustering algorithm to explore the legacy CUI and learn it while he is modeling it, gradually and iteratively. A review of the screen snapshots of MIRLYN revealed that its CUI style maintains useful information in the top rows of almost all screen snapshots. This suggests that the presentation space feature set of section 4.2 would be useful in clustering the snapshots of the input trace. On the other hand, different catalogs and indexes have very similar screen layouts for functionally similar screens. For example, the "Author Index" screens of different catalogs look very close to each other, and in some cases, the only difference is the catalog name. This suggests that the presentation space layout features of section 4.4 would not be very useful in clustering the snapshots. All the feature subsets of LeNDI and combinations of them were tried in clustering the snapshot set.

After about 10 rounds of setup, clustering and result review, the analyst reached a satisfactory partition that is near perfect in his judgement. The final setup used included one recognizer that was configured as in Table 5.7. A threshold of 100% was used. 88 clusters were discovered. The results were reviewed and it was discovered that:

- One cluster is redundant, i.e., it has few snapshots that should be in another cluster,
- The instances of two clusters are mixed together, and
- The instances of four other clusters are mixed together.

These errors were fixed manually. This required 5 MoJo Plus moves: 4 MM steps and one JOIN step. The result was 91 distinguished clusters, which are shown with their description and cardinality in Table 5.8. These clusters were the input to the classifier induction phase.

Feature	Description	Weight	Ignore if Empty
1-1	Encoding of the information at the snapshot periphery	20	N
1-2	Encoding of the start columns of all titles and codes	10	N
1-3	The text in the middle of the second non-blank row	30	Y
1-4	The text in the right of the second non-blank row	40	Y

Table 5.7. The Features Used for Setting up The Single-path Incremental Clustering Algorithm for MIRLYN Experiment 5.3.

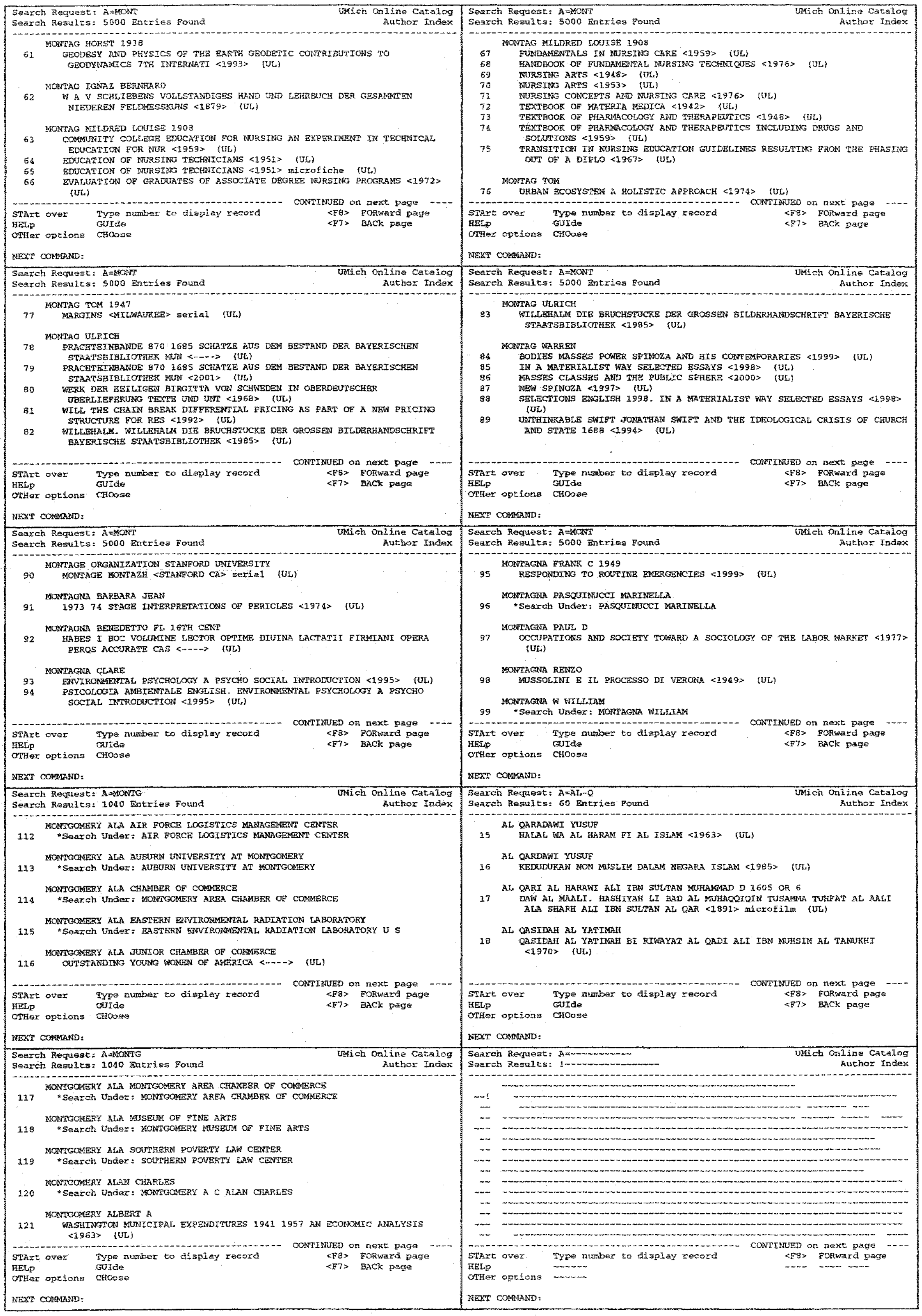
Id	Screen Description	Cr	Id	Screen Description	Cr
1	U of Michigan Libraries Main Menu	81	47	Renaissance Lit No Holdings Found	3
2	UMich Catalog Introduction	29	48	Renaissance Lit No Keyword Entry Found	10
3	UMich Catalog Author Guide	27	49	Africana Introduction	5
4	UMich Catalog Author Index	22	50	Africana Subject Guide	26
5	UMich Catalog Review Search List	12	51	Africana Subject Index	19
6	UMich Catalog Brief View	32	52	Africana Brief View	3
7	UMich Catalog Holdings Detail	5	53	Africana Long View	53
8	UMich Catalog Long View	120	54	Africana No Holdings Found	6
9	UMich Catalog No Title Entries Found	3	55	Africana Review Search List	7
10	UMich Catalog Other Options	8	56	Africana No Subject Entries Fou	4
11	UMich Catalog Title Guide	10	57	Africana Other options	1
12	UMich Catalog Title Index	12	58	Africana Title Index	22
13	UMich Catalog Explanation of MIRLYN	7	59	Africana Title Guide	10
14	UMich Catalog Explain Options	4	60	Africana No Title Entries Found	1
15	UMich Catalog Long View Help	1	61	Africana Holdings Detail	4
16	UMich Catalog Title Index Help	1	62	Electronic Resources Introduction	4
17	UMich Catalog No Author Entries Found	4	63	Electronic Resources Subject Guide	4
18	UMich Catalog Call Number Browse	17	64	Electronic Resources Subject Index	6
19	UMich Catalog Explain Call Number	4	65	Electronic Resources Long View	50
20	UMich Catalog Explain Catalog	8	66	Electronic Resources Holdings Detail	6
21	Course Reserve Search Menu	59	67	Electronic Resources Other Options	4
22	Course Reserve Index by Course	40	68	Electronic Resources Author Index	22
23	Course Reserve Index by Instructor	16	69	Electronic Resources Long View Help	3
24	Course Reserve Index by Title	10	70	Electronic Resources Author Guide	10
25	Course Reserve View Detail	46	71	Electronic Resources Explain Options	3
26	Course Reserve View Detail Help	3	72	Electronic Resources Explain Display	4
27	UMich Catalog Explain Author	3	73	Electronic Resources Review Search List	7
28	UMich Catalog Subject Index	12	74	Electronic Resources No Author Entry Fo	2
29	UMich Catalog No Subject Entries Found	1	75	Electronic Resources Title Index	4
30	UMich Catalog Subject Guide	13	76	Ohio St. Univ. Introduction	6
31	UMich Catalog Call Number Browse	2	77	Ohio St. Univ. News	3
32	Map Library Introduction	5	78	Ohio St. Univ. Other Options	4
33	Map Library Author Guide	24	79	Ohio St. Univ. Author Index	11
34	Map Library Author Index	20	80	Ohio St. Univ. Brief View	4
35	Map Library Brief View	20	81	Ohio St. Univ. Long View	48
36	Map Library Long View	20	82	Ohio St. Univ. Title Index	20
37	Map Library No Author Entries Found	3	83	Ohio St. Univ. Holdings Detail	12
38	Map Library No Review Search List	8	84	Ohio St. Univ. Review Search List	4
39	Map Library Heading Information	1	85	Ohio St. Univ. Holdings Detail Help	3
40	Map Library Title Guide	8	86	Ohio St. Univ. Explain Author	4
41	Map Library Title Index	19	87	UMich Catalog Subject Index Help	1
42	Renaissance Lit Introduction	11	88	Map Library No Title Entries Found	1
43	Renaissance Lit Keyword Index	17	89	Electronic Resources Title Guide	2
44	Renaissance Lit Brief View	5	90	Ohio St. Univ. Long View Help	2
45	Renaissance Lit Long View	33	91	UMich Catalog Holdings Detail Help	1
46	Renaissance Lit Other Options	2			

Table 5.8. Screen Descriptions and Cardinality for MIRLYN Experiment 5.3.

5.5.4.2 Classifier Induction

After fixing clustering results, the author asked LeNDI to generate a cluster signature for every cluster. Such a signature captures the commonality of the feature vectors and presentation spaces of the snapshots of a cluster in an artificial feature vector and an artificial presentation space. Each of Figures 5.10 to 5.12 shows 9 sample snapshots and the artificial presentations spaces of one of screens 4, 5 and 6 of Table 5.8, respectively. The artificial presentation space is at the right bottom of each figure. One can see that they nicely capture the common structure and content of their clusters.

To test these signatures, a test data set was recorded while the user performed more interaction with MIRLYN system. The set included 687 snapshots, in three traces of lengths 150, 256 and 281 snapshots. Next, LeNDI 's signature-based classifier was used to classify the snapshots of these traces. Successfully, LeNDI's signature-based classifier was able to correctly classify all the 687 snapshots.



```
Search Request: A=MONT                                UMich Online Catalog
Search Results: 5000 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTAG HORST 1938
  61    GEODESY AND PHYSICS OF THE EARTH GEODETIC CONTRIBUTIONS TO
         GEODYNAMICS 7TH INTERNATI <1993>  (UL)

     MONTAG IGNAZ BERNHARD
  62    W A V SCHLIEBENS VOLLSTANDIGES HAND UND LEHRBUCH DER GESAMMTEN
         NIEDEREN FELDMESSKUNS <1879>  (UL)

     MONTAG MILDRED LOUISE 1908
  63    COMMUNITY COLLEGE EDUCATION FOR NURSING AN EXPERIMENT IN TECHNICAL
         EDUCATION FOR NUR <1959>  (UL)
  64    EDUCATION OF NURSING TECHNICIANS <1951>  (UL)
  65    EDUCATION OF NURSING TECHNICIANS <1951> microfiche  (UL)
  66    EVALUATION OF GRADUATES OF ASSOCIATE DEGREE NURSING PROGRAMS <1972>
         (UL)
---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

```
Search Request: A=MONT                                UMich Online Catalog
Search Results: 5000 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTAG MILDRED LOUISE 1908
  67    FUNDAMENTALS IN NURSING CARE <1959>  (UL)
  68    HANDBOOK OF FUNDAMENTAL NURSING TECHNIQUES <1976>  (UL)
  69    NURSING ARTS <1948>  (UL)
  70    NURSING ARTS <1953>  (UL)
  71    NURSING CONCEPTS AND NURSING CARE <1976>  (UL)
  72    TEXTBOOK OF MATERIA MEDICA <1942>  (UL)
  73    TEXTBOOK OF PHARMACOLOGY AND THERAPEUTICS <1948>  (UL)
  74    TEXTBOOK OF PHARMACOLOGY AND THERAPEUTICS INCLUDING DRUGS AND
         SOLUTIONS <1959>  (UL)
  75    TRANSITION IN NURSING EDUCATION GUIDELINES RESULTING FROM THE PHASING
         OUT OF A DIPLO <1967>  (UL)

     MONTAG TOM
  76    URBAN ECOSYSTEM A HOLISTIC APPROACH <1974>  (UL)
---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

```
Search Request: A=MONT                                UMich Online Catalog
Search Results: 5000 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTAG TOM 1947
  77    MARGINS <MILWAUKEE> serial  (UL)

     MONTAG ULRICH
  78    PRACHTEINBANDE 870 1685 SCHATZE AUS DEM BESTAND DER BAYERISCHEN
         STAATSBIBLIOTHEK MUN <---->  (UL)
  79    PRACHTEINBANDE 870 1685 SCHATZE AUS DEM BESTAND DER BAYERISCHEN
         STAATSBIBLIOTHEK MUN <2001>  (UL)
  80    WERK DER HEILIGEN BIRGITTA VON SCHWEDEN IN OBERDEUTSCHER
         UBERLIEFERUNG TEXTE UND UNT <1968>  (UL)
  81    WILL THE CHAIN BREAK DIFFERENTIAL PRICING AS PART OF A NEW PRICING
         STRUCTURE FOR RES <1992>  (UL)
  82    WILLEHALM. WILLEHALM DIE BRUCHSTUCKE DER GROSSEN BILDERHANDSCHRIFT
         BAYERISCHE STAATSBIBLIOTHEK <1985>  (UL)

---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

```
Search Request: A=MONT                                UMich Online Catalog
Search Results: 5000 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTAG ULRICH
  83    WILLEHALM DIE BRUCHSTUCKE DER GROSSEN BILDERHANDSCHRIFT BAYERISCHE
         STAATSBIBLIOTHEK <1985>  (UL)

     MONTAG WARREN
  84    BODIES MASSES POWER SPINOZA AND HIS CONTEMPORARIES <1999>  (UL)
  85    IN A MATERIALIST WAY SELECTED ESSAYS <1998>  (UL)
  86    MASSES CLASSES AND THE PUBLIC SPHERE <2000>  (UL)
  87    NEW SPINOZA <1997>  (UL)
  88    SELECTIONS ENGLISH 1998. IN A MATERIALIST WAY SELECTED ESSAYS <1998>
         (UL)
  89    UNTHINKABLE SWIFT JONATHAN SWIFT AND THE IDEOLOGICAL CRISIS OF CHURCH
         AND STATE 1688 <1994>  (UL)

---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

```
Search Request: A=MONT                                UMich Online Catalog
Search Results: 5000 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTAGE ORGANIZATION STANFORD UNIVERSITY
  90    MONTAGE MONTAZH <STANFORD CA> serial  (UL)

     MONTAGNA BARBARA JEAN
  91    1973 74 STAGE INTERPRETATIONS OF PERICLES <1974>  (UL)

     MONTAGNA BENEDETTO FL 16TH CENT
  92    HABES I HOC VOLUMINE LECTOR OPTIME DIUINA LACTATII FIRMIANI OPERA
         PERQS ACCURATE CAS <---->  (UL)

     MONTAGNA CLARE
  93    ENVIRONMENTAL PSYCHOLOGY A PSYCHO SOCIAL INTRODUCTION <1995>  (UL)
  94    PSICOLOGIA AMBIENTALE ENGLISH. ENVIRONMENTAL PSYCHOLOGY A PSYCHO
         SOCIAL INTRODUCTION <1995>  (UL)
---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

```
Search Request: A=MONT                                UMich Online Catalog
Search Results: 5000 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTAGNA FRANK C 1949
  95    RESPONDING TO ROUTINE EMERGENCIES <1999>  (UL)

     MONTAGNA PASQUINUCCI MARINELLA
  96   *Search Under: PASQUINUCCI MARINELLA

     MONTAGNA PAUL D
  97    OCCUPATIONS AND SOCIETY TOWARD A SOCIOLOGY OF THE LABOR MARKET <1977>
         (UL)

     MONTAGNA RENZO
  98    MUSSOLINI E IL PROCESSO DI VERONA <1949>  (UL)

     MONTAGNA W WILLIAM
  99   *Search Under: MONTAGNA WILLIAM
---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

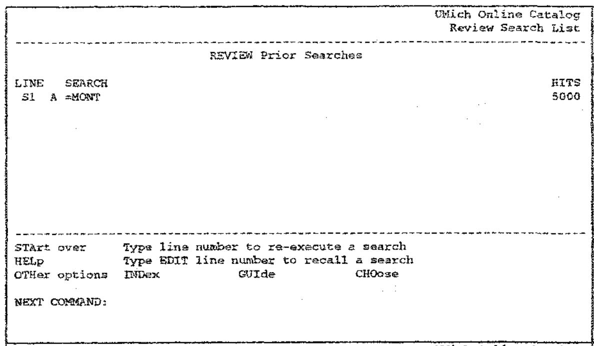
```
Search Request: A=MONTG                               UMich Online Catalog
Search Results: 1040 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTGOMERY ALA AIR FORCE LOGISTICS MANAGEMENT CENTER
 112   *Search Under: AIR FORCE LOGISTICS MANAGEMENT CENTER

     MONTGOMERY ALA AUBURN UNIVERSITY AT MONTGOMERY
 113   *Search Under: AUBURN UNIVERSITY AT MONTGOMERY

     MONTGOMERY ALA CHAMBER OF COMMERCE
 114   *Search Under: MONTGOMERY AREA CHAMBER OF COMMERCE

     MONTGOMERY ALA EASTERN ENVIRONMENTAL RADIATION LABORATORY
 115   *Search Under: EASTERN ENVIRONMENTAL RADIATION LABORATORY U S

     MONTGOMERY ALA JUNIOR CHAMBER OF COMMERCE
 116    OUTSTANDING YOUNG WOMEN OF AMERICA <---->  (UL)

---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

```
Search Request: A=AL-Q                                UMich Online Catalog
Search Results: 60 Entries Found                              Author Index
--------------------------------------------------------------------------
     AL QARADAWI YUSUF
  15    HALAL WA AL HARAM FI AL ISLAM <1963>  (UL)

     AL QARDAWI YUSUF
  16    KEDUDUKAN NON MUSLIM DALAM NEGARA ISLAM <1985>  (UL)

     AL QARI AL HARAWI ALI IBN SULTAN MUHAMMAD D 1605 OR 6
  17    DAW AL MAALI. HASHIYAH LI BAD AL MUHAQQIQIN TUSAMMA TUHFAT AL AALI
         ALA SHARH ALI IBN SULTAN AL QAR <1891> microfilm  (UL)

     AL QASIDAH AL YATIMAH
  18    QASIDAH AL YATIMAH BI RIWAYAT AL QADI ALI IBN MUHSIN AL TANUKHI
         <1970>  (UL)

---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

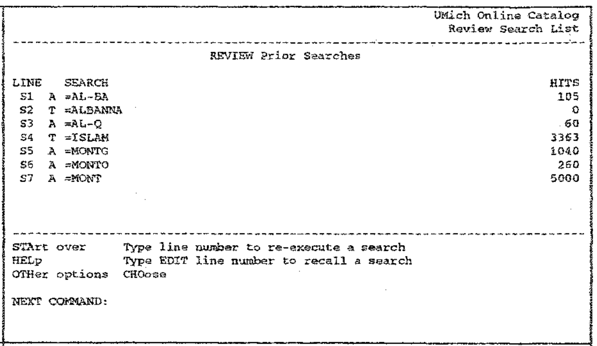
```
Search Request: A=MONTG                               UMich Online Catalog
Search Results: 1040 Entries Found                            Author Index
--------------------------------------------------------------------------
     MONTGOMERY ALA MONTGOMERY AREA CHAMBER OF COMMERCE
 117   *Search Under: MONTGOMERY AREA CHAMBER OF COMMERCE

     MONTGOMERY ALA MUSEUM OF FINE ARTS
 118   *Search Under: MONTGOMERY MUSEUM OF FINE ARTS

     MONTGOMERY ALA SOUTHERN POVERTY LAW CENTER
 119   *Search Under: SOUTHERN POVERTY LAW CENTER

     MONTGOMERY ALAN CHARLES
 120   *Search Under: MONTGOMERY A C ALAN CHARLES

     MONTGOMERY ALBERT A
 121    WASHINGTON MUNICIPAL EXPENDITURES 1941 1957 AN ECONOMIC ANALYSIS
         <1963>  (UL)
---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            GUIde                                  <F7>  BACk page
OTHer options   CHOose

NEXT COMMAND:
```

```
Search Request: A=-----------                         UMich Online Catalog
Search Results: !---------------                              Author Index
--------------------------------------------------------------------------
     ---------------------------------------------------
 --!    -----------------------------------------------------------------
 --      --------------------------------------------------- ------- ---
 --    -------------------------------------------------- ------ ----- ----
 ---   ---------------------------------------------------------------------
 --    -----------------------------------------------------------------
 --    ------------------------------------------------------------------
 ---   ---------------------------------------------------------------------
 --    ----------------------------------------------------------------
 --    --------------------------------------------------------------------
 ---   ---------------------------------------------------------------------
 --    ---------------------------------------------------------------------
 --    --------------------------------------------------------------------
 ---   --------------------------------------------------------------------
 --     -------------------------------------------------------------  ----
---------------------------------------------- CONTINUED on next page ----
STArt over      Type number to display record          <F8>  FORward page
HELp            ------                                 ----  ----- ----
OTHer options   ------

NEXT COMMAND:
```

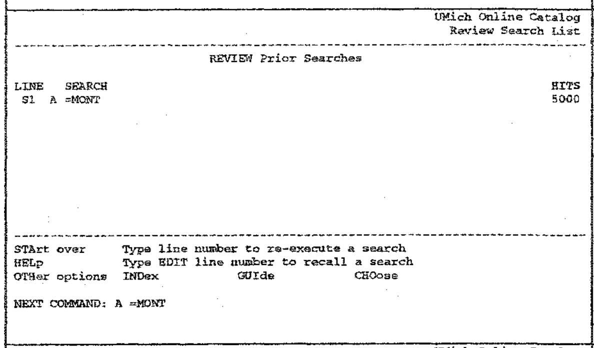
Figure 5.10. The Signature and Some Snapshots of Cluster 4 of Experiment 5.3.



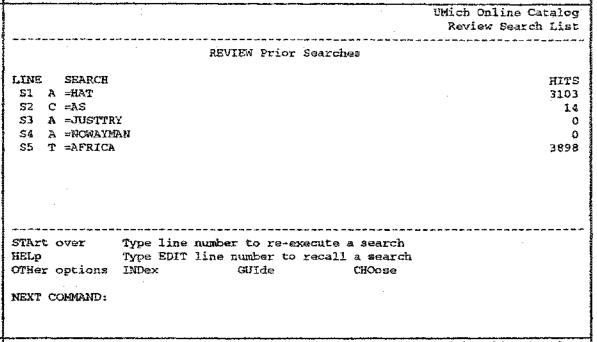
```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =MONT                                                  5000

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   INDex          GUIde          CHOose

NEXT COMMAND:
```



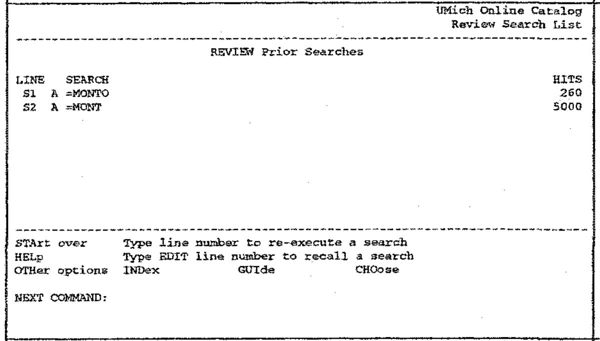
```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =AL-BA                                                  105
 S2  T =ALBANNA                                                  0
 S3  A =AL-Q                                                    60
 S4  T =ISLAM                                                 3363
 S5  A =MONTG                                                 1040
 S6  A =MONTO                                                  260
 S7  A =MONT                                                  5000

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   CHOose

NEXT COMMAND:
```



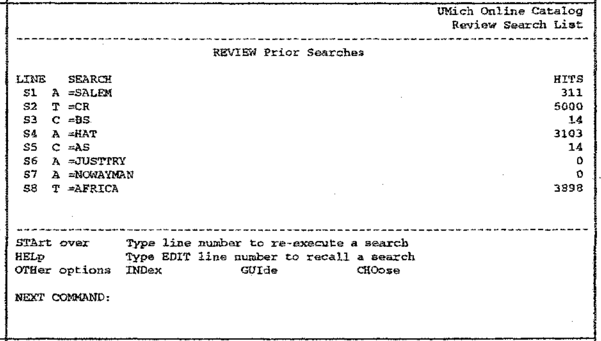
```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =MONT                                                  5000

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   INDex          GUIde          CHOose

NEXT COMMAND: A =MONT
```



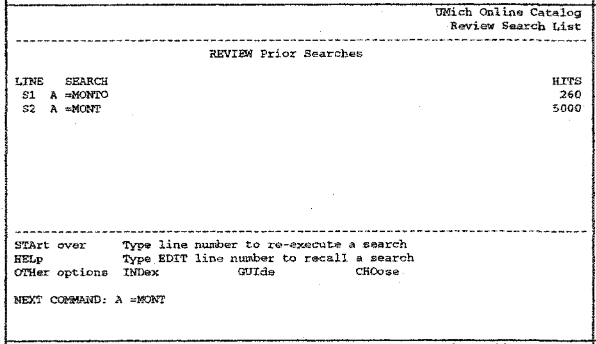
```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =HAT                                                   3103
 S2  C =AS                                                      14
 S3  A =JUSTTRY                                                  0
 S4  A =NOWAYMAN                                                 0
 S5  T =AFRICA                                                3898

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   INDex          GUIde          CHOose

NEXT COMMAND:
```



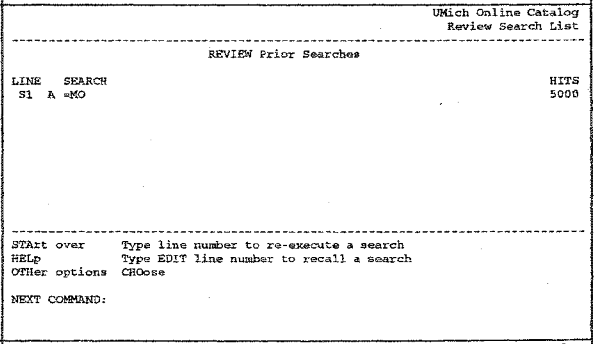
```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =MONTO                                                  260
 S2  A =MONT                                                  5000

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   INDex          GUIde          CHOose

NEXT COMMAND:
```



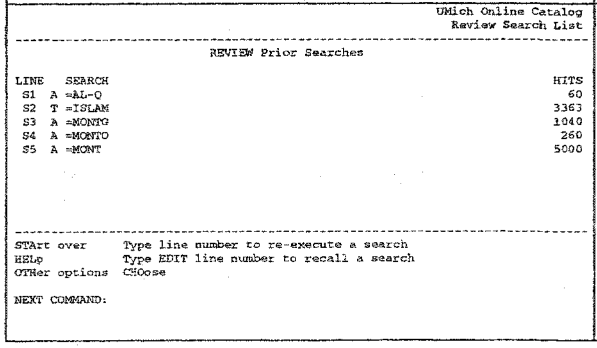
```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =SALEM                                                  311
 S2  T =CR                                                    5000
 S3  C =BS                                                      14
 S4  A =HAT                                                   3103
 S5  C =AS                                                      14
 S6  A =JUSTTRY                                                  0
 S7  A =NOWAYMAN                                                 0
 S8  T =AFRICA                                                3898

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   INDex          GUIde          CHOose

NEXT COMMAND:
```



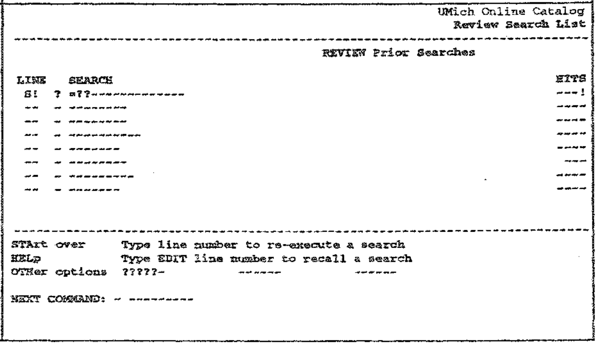
```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =MONTO                                                  260
 S2  A =MONT                                                  5000

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   INDex          GUIde          CHOose

NEXT COMMAND: A =MONT
```



```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =MO                                                    5000

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   CHOose

NEXT COMMAND:
```

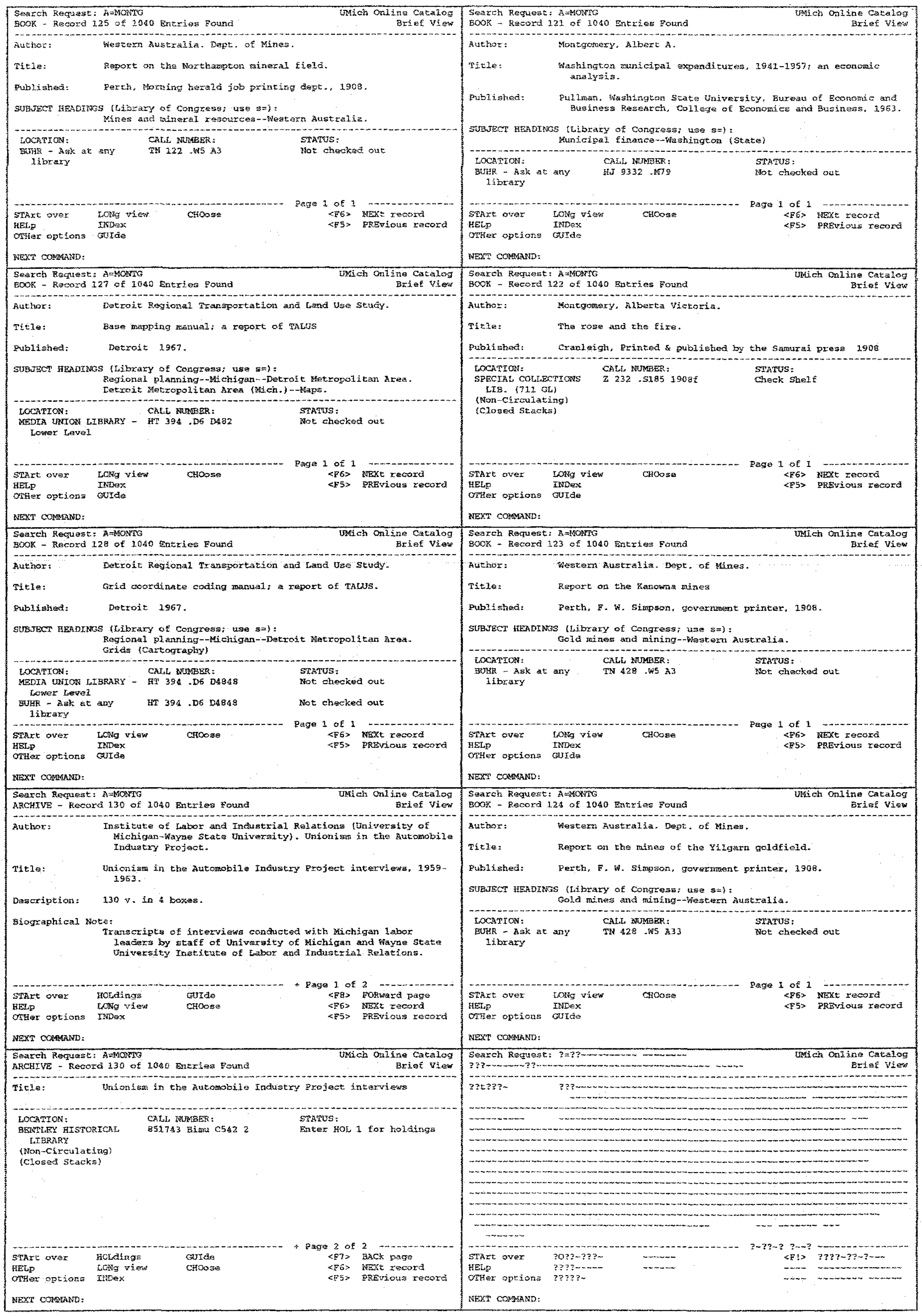


```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S1  A =AL-Q                                                    60
 S2  T =ISLAM                                                 3363
 S3  A =MONTG                                                 1040
 S4  A =MONTO                                                  260
 S5  A =MONT                                                  5000

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   CHOose

NEXT COMMAND:
```



```
                                              UMich Online Catalog
                                                Review Search List
------------------------------------------------------------------
                   REVIEW Prior Searches

LINE   SEARCH                                                 HITS
 S!  ? =??-------------                                       ---!
 --  - ---------                                              ----
 --  - ---------                                              ----
 --  - -----------                                            ----
 --  - --------                                               ----
 --  - ---------                                               ---
 --  - ----------                                             ----
 --  - --------                                               ----

------------------------------------------------------------------
STArt over      Type line number to re-execute a search
HELp            Type EDIT line number to recall a search
OTHer options   ?????-         ------         ------

NEXT COMMAND: - ---------
```

Figure 5.11. The Signature and Some Snapshots of Cluster 5 of Experiment 5.3.



```
Search Request: A=MONTG                                   UMich Online Catalog
BOOK - Record 125 of 1040 Entries Found                             Brief View
------------------------------------------------------------------------------
Author:          Western Australia. Dept. of Mines.

Title:           Report on the Northampton mineral field.

Published:       Perth, Morning herald job printing dept., 1908.

SUBJECT HEADINGS (Library of Congress; use s=):
                 Mines and mineral resources--Western Australia.
------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 BUHR - Ask at any        TN 122 .W5 A3              Not checked out
   library



---------------------------------------------- Page 1 of 1 ---------------
STArt over     LONg view       CHOose                    <F6>  NEXt record
HELp           INDex                                     <F5>  PREvious record
OTHer options  GUIde

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
BOOK - Record 121 of 1040 Entries Found                             Brief View
------------------------------------------------------------------------------
Author:          Montgomery, Albert A.

Title:           Washington municipal expenditures, 1941-1957; an economic
                   analysis.

Published:       Pullman, Washington State University, Bureau of Economic and
                   Business Research, College of Economics and Business, 1963.

SUBJECT HEADINGS (Library of Congress; use s=):
                 Municipal finance--Washington (State)
------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 BUHR - Ask at any        HJ 9332 .M79               Not checked out
   library

---------------------------------------------- Page 1 of 1 ---------------
STArt over     LONg view       CHOose                    <F6>  NEXt record
HELp           INDex                                     <F5>  PREvious record
OTHer options  GUIde

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
BOOK - Record 127 of 1040 Entries Found                             Brief View
------------------------------------------------------------------------------
Author:          Detroit Regional Transportation and Land Use Study.

Title:           Base mapping manual; a report of TALUS

Published:        Detroit  1967.

SUBJECT HEADINGS (Library of Congress; use s=):
                 Regional planning--Michigan--Detroit Metropolitan Area.
                 Detroit Metropolitan Area (Mich.)--Maps.
------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 MEDIA UNION LIBRARY -    HT 394 .D6 D482            Not checked out
   Lower Level


---------------------------------------------- Page 1 of 1 ---------------
STArt over     LONg view       CHOose                    <F6>  NEXt record
HELp           INDex                                     <F5>  PREvious record
OTHer options  GUIde

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
BOOK - Record 122 of 1040 Entries Found                             Brief View
------------------------------------------------------------------------------
Author:          Montgomery, Alberta Victoria.

Title:           The rose and the fire.

Published:       Cranleigh, Printed & published by the Samurai press  1908
------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 SPECIAL COLLECTIONS      Z 232 .S185 1908f          Check Shelf
   LIB. (711 GL)
 (Non-Circulating)
 (Closed Stacks)




---------------------------------------------- Page 1 of 1 ---------------
STArt over     LONg view       CHOose                    <F6>  NEXt record
HELp           INDex                                     <F5>  PREvious record
OTHer options  GUIde

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
BOOK - Record 128 of 1040 Entries Found                             Brief View
------------------------------------------------------------------------------
Author:          Detroit Regional Transportation and Land Use Study.

Title:           Grid coordinate coding manual; a report of TALUS.

Published:        Detroit  1967.

SUBJECT HEADINGS (Library of Congress; use s=):
                 Regional planning--Michigan--Detroit Metropolitan Area.
                 Grids (Cartography)
------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 MEDIA UNION LIBRARY -    HT 394 .D6 D4848           Not checked out
   Lower Level
 BUHR - Ask at any        HT 394 .D6 D4848           Not checked out
   library
---------------------------------------------- Page 1 of 1 ---------------
STArt over     LONg view       CHOose                    <F6>  NEXt record
HELp           INDex                                     <F5>  PREvious record
OTHer options  GUIde

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
BOOK - Record 123 of 1040 Entries Found                             Brief View
------------------------------------------------------------------------------
Author:          Western Australia. Dept. of Mines.

Title:           Report on the Kanowna mines

Published:       Perth, F. W. Simpson, government printer, 1908.

SUBJECT HEADINGS (Library of Congress; use s=):
                 Gold mines and mining--Western Australia.
------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 BUHR - Ask at any        TN 428 .W5 A3              Not checked out
   library


---------------------------------------------- Page 1 of 1 ---------------
STArt over     LONg view       CHOose                    <F6>  NEXt record
HELp           INDex                                     <F5>  PREvious record
OTHer options  GUIde

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
ARCHIVE - Record 130 of 1040 Entries Found                          Brief View
------------------------------------------------------------------------------
Author:          Institute of Labor and Industrial Relations (University of
                   Michigan-Wayne State University). Unionism in the Automobile
                   Industry Project.

Title:           Unionism in the Automobile Industry Project interviews, 1959-
                   1963.

Description:     130 v. in 4 boxes.

Biographical Note:
                 Transcripts of interviews conducted with Michigan labor
                   leaders by staff of University of Michigan and Wayne State
                   University Institute of Labor and Industrial Relations.

-------------------------------------------- + Page 1 of 2 -------------
STArt over     HOLdings        GUIde                     <F8>  FORward page
HELp           LONg view       CHOose                    <F6>  NEXt record
OTHer options  INDex                                     <F5>  PREvious record

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
BOOK - Record 124 of 1040 Entries Found                             Brief View
------------------------------------------------------------------------------
Author:          Western Australia. Dept. of Mines.

Title:           Report on the mines of the Yilgarn goldfield.

Published:       Perth, F. W. Simpson, government printer, 1908.

SUBJECT HEADINGS (Library of Congress; use s=):
                 Gold mines and mining--Western Australia.
------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 BUHR - Ask at any        TN 428 .W5 A33             Not checked out
   library


---------------------------------------------- Page 1 of 1 ---------------
STArt over     LONg view       CHOose                    <F6>  NEXt record
HELp           INDex                                     <F5>  PREvious record
OTHer options  GUIde

NEXT COMMAND:
```

```
Search Request: A=MONTG                                   UMich Online Catalog
ARCHIVE - Record 130 of 1040 Entries Found                          Brief View
------------------------------------------------------------------------------
Title:           Unionism in the Automobile Industry Project interviews

------------------------------------------------------------------------------
 LOCATION:                CALL NUMBER:               STATUS:
 BENTLEY HISTORICAL       851743 Bimu C542 2         Enter HOL 1 for holdings
   LIBRARY
 (Non-Circulating)
 (Closed Stacks)






-------------------------------------------- + Page 2 of 2 -------------
STArt over     HOLdings        GUIde                     <F7>  BACk page
HELp           LONg view       CHOose                    <F6>  NEXt record
OTHer options  INDex                                     <F5>  PREvious record

NEXT COMMAND:
```

```
Search Request: ?=??---------- --------                   UMich Online Catalog
???-------??---------------------------------- -----                Brief View
------------------------------------------------------------------------------
??t???~          ???----------------------------------------------------------
                   ----------------------------------------- -----------------
------------------------------------------------------------------------------
-----------      --------------------------------------------------------- ---
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
-------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
------------------------------------------------------------------------------
 ---------------------------------------------          --- -------- ---
   -------
---------------------------------------------- ?~??~? ?~~? ---------------
STArt over     ?O??~???~       ------                    <F!>  ????~??~?~--
HELp           ????-----       ------                    ----  --------------
OTHer options  ?????~                                    ----  --------- ------

NEXT COMMAND:
```

Figure 5.12. The Signature and Some Snapshots of Cluster 6 of Experiment 5.3.

5.5.4.3 Transition Modeling

Next, transition modeling was performed on the input interaction trace. 369 transitions were discovered. Some of these transitions are shown in Table 5.9. The transition *<i><action><j>* is represented by the action model *action* at row *i* and column *j*. A '*' means a mandatory keyword, argument or option, i.e., something has to be input in this location. A '[*]' means an optional input. @e is Enter key and @*n* is the control key PF*n*. A '!' means a numerical input is mandatory. Some action models are too general due to lack of examples, i.e., LeNDI could not discover the specific keywords or options of the action, although some exist, due to lack of enough examples. In this case, LeNDI takes a safe route by assuming a too general model. Such action models are shown with gray backgrounds. Some action models are too narrow, which means that the model would not generalize probably and it overfits the examples available in the trace. This happens when the examples of this action were too similar to each other. Such models are shown with white font and black background. The locations (x,y coordinates or range) of action models are omitted to spare space in Table 5.9.

The entry point to MIRLYN is screen 1 (U of Michigan Libraries Main Menu). This menu is 14 screens long, with each of them presenting new options to the user. In other words, screen 1 has 14 versions. The user may access more instances of this screen by moving forward by pressing Enter or PF8, typing "*remote*" followed by Enter or typing an undefined string followed by Enter. The user can move to screen 2 (UMich Catalog Introduction) and open University of Michigan catalog by just pressing Enter (cell 1,2). From screen 2, the user can open screen 3 (UMich Author Guide) or screen 11 (UMich Title Guide) by issuing the command "*a = *@e*" or "*t* = *@e", respectively, as in row 2. These two commands are the catalog commands for searching for a specific author or title. Similarly, the user can move to other screens as her/his task needs.

One can see that some columns have similar action models, suggesting that the corresponding screen can be reached using the same action from only some specific screens. For example, screen 7 (UMich Catalog Holdings Detail) can be accessed only by typing "*hold*" or "*hol*" and pressing Enter from screens 6 (UMich Catalog Brief View) or 8 (UMich Catalog Long View). Another example is screen 10 (UMich Catalog Other Options). It can be accessed only by typing "*oth*" and pressing Enter. However, many action models in column 10 have gray background due to lack of examples, and hence, they are too general and do not show the specific keyword ("*oth*"). Too general models occur due to lack of examples and are not a problem. This is because lack of examples means that the action occurred a rare navigation sequence. Such sequences would not be discovered as an interaction pattern. If needed, a too general action model can be made more specific manually.

An example of an overfitting model is the transition from screen 5 to screen 5, via the action model "edit s1@e". The *edit* command allows the system user to edit one of the previous searches in the open library catalog performed previously. Since many of the examples had the user revising the first line in his search history (line s1), LeNDI mistakenly assumed that "s1" is a possible argument of the command. This problem can be fixed either by collecting more examples of the same action or by changing the model manually.

	U of Michigan Libraries Main Menu	UMich Catalog Introduction	UMich Catalog Author Guide	UMich Catalog Author Index	UMich Catalog Review Search List	UMich Catalog Brief View	UMich Catalog Holdings Detail	UMich Catalog Long View	UMich Catalog No Title Entries Found	UMich Catalog Other Options	UMich Catalog Title Guide	UMich Catalog Title Index	UMich Catalog Explanation of MIRLYN
Id	1	2	3	4	5	6	7	8	9	10	11	12	13
1	remote@e [*]@e @8	@e											
2	sta@e *@e	[*]@e @7	a = *@e		rev@e						t = *@e		
3	*@e		@e	in@e ind@e !@e	rev@e								
4			*[*][*]@e	@e	rev@e	!@e		!@e	* * *@e	*@e			
5			a = *@e *@e		edit s1@e edit *@e				* * *@e		* * *@e		
6				a = *[*]@e		@6 @7 @8 nex@e [*]@e	hol@e	1@e lon@e		*@e			
7						*@e		*@e @6		*@e			
8	cho@e sta@e			ind@e		bri@e	hol@e hold@e	[*][=][*]@e @e @5 @6 @8		oth@e	*@e	ind@e	
9			* * *@e										
10											t = *@e		
11												!@e *@e	
12					*@e	!@e		!@e			*@e	@e	
13			* * *@e		*@e								[*]@e @8
14												ex mirlyn@e exp mirlyn@e	
15								@e					
16												@e	
17	*@e				*@e					*@e			
18						*@e		12@e		*@e			
19										*@e			
20													

Table 5.9. Some of The Transition Models Built by LeNDI for The MIRLYN Trace of Experiment 5.3.

5.6 Discussion

This chapter presented the interaction-based legacy UI reverse engineering process developed in CelLEST project. The process builds a behavioral model for a character-based legacy user interface using traces of interaction with the legacy system, recorded while its users are performing their regular activities using the legacy system. The model is in the form of a state-transition model. Its nodes represent the behavioral states or screens of the legacy system. Its edges represent the user actions necessary to cause a transition from the source screen where the action took place to a destination screen. The two substasks in building this model are identifying its nodes and its edges. The first is accomplished by clustering similar screen snapshots together automatically or semi-automatically, verifying clustering results, and then inducing a classifier that is able to generalize clustering results by classifying new snapshots to one of the existing clusters. The second process involves analyzing the available instances of every transaction between two screens, assuming that all the user keystrokes done to initiate such a transition are instances of the same user action. This analysis leads to identifying what is common in these instances and what is variable in terms of their textual contents and locations on the source screen. The following discussion elaborates on the strengths, limitations and possible enhancements of this work.

5.6.1 Strengths

This subsection presents the strengths of the CUI behavioral modeling process presented in this chapter. It discusses how this process advances current modeling practices, while requiring lower skills and less time, effort and cost.

5.6.1.1 A Coherent Automated CUI Behavior Modeling Process

The novel coherent CUI behavior modeling process of LeNDI supersedes the manual practices used currently in industry, which are described in fair detail in section 2.2. These practices focus on manual modeling of either some interesting parts of a given legacy CUI or the user tasks that will be reengineered, from scratch.

In these practices, using a simple or rich pattern language, an analyst manually defines classifiers that are able to classify some of the snapshots of the system in hand to one of a set of classes, each represents a legacy system screen. Since the analyst is usually unfamiliar with the system in hand, s/he needs the aid of some expert users of the system. Also s/he needs to familiarize her/himself with the system via reading the system documents, if any, talking with and observing the system users, trial and error, reading help screens, etc. Then s/he needs to figure out how many distinct screens or states are there in order to infer classifiers for them. Classifiers are logical combinations of patterns described in the given pattern language. For example, patterns can represent the existence or absence of a certain text at a certain location or within a range on the screen. To do the modeling task, the analyst goes through many sample snapshots of each screen under analysis trying to discover what is common on them and what differentiates them from other instances of other screens. This is done by visually inspecting the snapshots and finding one pattern or a combination of patterns that distinguishes the snapshots. This is quite a labor-intensive and time-consuming job. Moreover, due to the limited set of features offered by pattern languages, some times it is very hard to construct a logical expression of patterns to distinguish the instances of a screen and may require forming quite complex patterns.

To model the possible transitions among the screens of a legacy system, the analyst needs to see different instances of each transition, try them and familiarize him/herself with the interaction style adopted in the legacy CUI. By analyzing these instances, s/he has to infer a model of each transition that describes what keywords, arguments and options are needed/possible for this transition and what are the possible variants for each of them. Also, her/his manual analysis should discover how many pieces of data are needed for the transition and whether they are optional or mandatory. Finally, s/he needs to figure out where on the screens all these pieces take place.

LeNDI provides an alternative coherent semi-automated behavior modeling process that eliminates the need for tedious error-prone manual model building. It pushes interaction-based legacy UI reengineering forward and significantly advances its current practices making it a favorable solution when code reengineering is not essential. By relieving the analyst from manual piece by piece model building, LeNDI saves time, effort and cost.

While this process relies on inferring most of the elements of the required behavioral model almost automatically, it leaves room for user feedback to revise the generated model and overrule LeNDI's decisions.

5.6.1.2 Low Skills

LeNDI requires lower skills than current manual practices. To use LeNDI, moderate analysis skills and fair understanding of the system under analysis is needed. Current practices require solid software development and programming skills and the aid of expert users of the legacy system. This is because LeNDI infers many pieces of information automatically and asks the analyst just for verification and feedback. While in current practices, all information is extracted manually.

5.6.1.3 Comprehensibility of the Results

The state-transition model produced by LeNDI is intuitively understandable, with little legacy system user experience. Hence, verifying the model and giving feedback is easy. This is especially true due to the coherent modeling approach used in LeNDI as opposed to the segmented approach followed in current practices and because of the tool support for visualization, offered by QandA [Vij02].

5.6.1.4 Flexibility and Extensibility

LeNDI's behavior modeling process is flexible to possible changes in the legacy system CUI. Minor changes can be done manually to the generated state-transition model and major ones can be done by recording new traces of interaction and partially redoing the modeling process.

LeNDI employs two different clustering algorithms that require different levels of familiarity with the legacy CUI and different inputs. It also employs two different classifier induction methods. Thus, it offers flexibility in choosing the right methods based on the legacy CUI in hand and the judgement of the analyst.

Additionally, the CUI modeling process is open to improvements by adding new features, clustering algorithms and/or classifier induction algorithms. Since the generated state-transition model is stored in an MS Access database, it would be possible to query it using SQL about possible navigation paths from a screen to another in order to build a navigation plan for a novel task model, or other reasons.

5.6.2 Limitations

Interaction-based legacy CUI behavior modeling in LeNDI has some limitations in terms of the accuracy and completeness of the model built and the necessity of some human setup and feedback. These limitations are detailed below.

5.6.2.1 Model Completeness and Classifier Accuracy

Two important questions should arise after presenting the experiments and evaluation of section 5.5. These questions are 1) how complete is the state-transition model produced? and 2) how accurate is the classifier induced to classify new snapshots as instances of exiting nodes or states? There is no straightforward answer to both questions as both measures rely heavily on the amount and quality of input data. Therefore, instead of giving a precise answer to these questions, the following elaborates on what factors affect these two measures.

First, model completeness depends on the coverage of the data collected. It is obvious that LeNDI would not be able to model states (screens) that were never accessed while recording the interaction traces, and hence no sample snapshots of them were available for clustering. The same applies to edges (transitions). However, a complete model of the entire legacy UI is not really of interest. Instead, "enough" modeling is what is needed. This means producing a model that covers the services of the legacy CUI that would be subject to reengineering, and the necessary related screens, e.g., help screens, messages screens, etc.

Second, the accuracy of the classifier, and consequently the ability of the new interface front-end to monitor and control the state of the underlying legacy application, depends on two factors. First, it is important that enough examples of all screens of the legacy interface have been recorded. Since the trace-recording emulator is not intrusive, a large number of emulators, installed on the terminals of a variety of legacy users for sufficiently long time should result in long and sufficiently representative traces. However for a given screen, "sufficient" is a function of its feature vector, content dynamics and its similarity to other screens.

Another factor affecting the classifier accuracy is the quality of the partition produced by the clustering process. This is why the clustering process is interactive and can be guided by the LeNDI analyst. Then, the clusters can be reviewed and incorrectly clustered snapshots can be identified. This process continues until all errors are eliminated. By analogy, the accuracy of transition modeling depends on having enough representing examples of each modeled transition. Lack of enough examples or having very similar examples may lead to too general or too specific models, respectively. Therefore, transition modeling is open to user revisions to fix the overfitting models and to specify the too general ones, if needed.

5.6.2.2 User Feedback Is Necessary

Albeit mostly automatic, some user input is still required in the current legacy CUI behavior modeling process of LeNDI. Complete automation of the reverse engineering process is not possible due to the variety of practices used in designing legacy CUIs. There will always be a need for some user feedback to complete the UI model. Smarter feature sets and better clustering and classifier induction methods can reduce the user feedback. The single-path incremental clustering algorithm relies on a good human setup of its parameters, while the top-down algorithm has minimized the needed input to only one number, the estimated number of clusters (or the maximum internal cluster incoherence threshold). But in both cases, clustering is done iteratively and a few rounds of clustering/results review are usually needed. And ultimately after clustering, user feedback is needed in the form of result review/revision before generating a classifier. This is to verify model correctness since there is some judgement needed for the modeling process. For example, on some screens, one may issue the wrong command or pass the wrong piece of data and as a result s/he receives what seems to be an instance of the same screen with an error message. Should the snapshot with the error message be considered an instance of the original one or an instance of a separate state? Does it exhibit a different behavior than the original one? These are questions whose answers need the judgement of the analyst.

5.6.3 Future Enhancements

In the following I include some of the areas where legacy CUI behavior modeling can be improved and enhanced.

5.6.3.1 Feature Selection for Clustering

Currently the LeNDI analyst has to decide which features to use for clustering snapshots using the single-path incremental clustering algorithm. On the other hand, s/he has no choice on which binary features to include in decision tree building using the top-down algorithm. This is because one of the motives behind developing this algorithm is relive the user from having to decide which features to use for clustering and leaving it to the algorithm to decide which feature to use for splitting at each decision tree node. There is a body of work on feature selection in machine learning [BL97, KS96] and a number of other areas. The main idea is that discovery and removal of irrelevant and redundant features with respect to a given data set can lead to more accurate results in clustering and classifier induction. In the problem in hand, this means less human input as well.

5.6.3.2 Enhancing Clustering and Classifier Induction

Currently, the LeNDI analyst chooses which clustering and classification methods to use for a given system. S/he can alternate between methods by using one of them and then switching to the other. However, currently, LeNDI does not allow hybrid clustering or classification by combining results from both clustering algorithms or both classification algorithms. This idea is worthy of investigation for potential performance improvement. During clustering phase and before user correction of clustering mistakes, it would be possible to apply both algorithms simultaneously and assign more confidence to the results that they both generated, i.e., the area where both produced partitions overlap. The same idea can be used for classifiers' decisions. One can generate both available classifiers and use them to classify a new snapshot simultaneously. In this case, more confidence should be given to the decision shared by both classifiers.

Another future improvement is adding a measure of matching strength for the signature-based classifier to use in case of more than one match due to loose signatures. It is possible also to add a signature analysis heuristic to discover loose signatures and report them to the analyst at design time and suggest switching to the decision tree classifier to identify new snapshots of the clusters with loose signatures.

Another future possible improvement is integrating more clustering algorithms and classifier induction algorithms to LeNDI to allow more choice for the analyst, depending on the system in hand. Also, it is possible to add simple pattern definition capabilities to LeDNI so that the user can create his own signature pattern if he wishes so for some screens.

5.6.3.3 Enhancing Action Modeling

The current version LeNDI has focused on systems adopting a combination of function key and command-driven interaction style, which is a frequently occurring combination. The future version of LeNDI will cover other forms of interaction, particularly, form-filling and menu selection.

Additional enhancements can include transition generalization. In many systems there are user actions that are available on many or most of the legacy UI screens, e.g., invoking a help screen, returning to a main menu or quitting the system. LeNDI would model such an action only as part of the transitions that has instances in the recorded traces that include this action. In other words, if it is possible to invoke a help screen from, say, 50 possible screens but there are only instances for invoking it from 5 screens, then LeNDI will model only these five transitions. It would be possible to use some document layout and content analysis methods, possibly with some user input, to compare areas on different screens that describe available standard user actions and use the results to generalize a transition. This means declaring the action of that transition a global action that is permissible on any screen with certain characteristics despite that there is no record of that action performed on an instance of the screen.

Chapter Six

Mining Interaction Traces for Patterns of Frequent User Tasks

Chapter 5 detailed the process of building behavior models for legacy system CUIs in the form of state-transition models. The next step in the CUI reverse engineering process is to discover what services of the legacy CUI are being used, or from a user's perspective, what frequent tasks the users execute through the legacy CUI. This is represented by task T2 in Figure 3.1. LeNDI discovers these frequent tasks in the form of frequent segments of interaction with the legacy CUI, or as we call them "interaction patterns". Each pattern is a hypothesis of a user task interesting enough to appear frequently in the traces. LeNDI analyst needs to verify these hypotheses. Semi-automatically in task T3, the forward engineering tool of CelLEST, Mathaino [Kap01], augments each verified interaction pattern with the information exchange that occurs during its execution. Thus, each interaction pattern provides the basis for a task model that is used in generating abstract GUI specifications in task T4. At runtime, these task models are used by the XHTML or WML front-ends generated in task T5 to execute the corresponding user task, feed the legacy application with user inputs, collect the required outputs and present them to the user through the new front-end.

Current industrial practices, as described in section 2.2.3, do not support automatic discovery of frequent user tasks. Instead, they adopt a manual modeling process during which, an analyst and an expert user sit together and manually define the navigation sequence to take place for every user task. So, given the state-transition model of the legacy system, they need to identify the main navigation path and any alternative paths for the task in terms of the starting screen and the sequence of screens to be accessed to perform the task. LeNDI automates this process by mining the recorded interaction traces for interaction patterns. The LeNDI analyst has the freedom to accept, reject or modify the discovered interaction patterns after reviewing them. This is done to verify that each pattern represents an actual user task of interest and faithfully represents the navigational path traversed to execute this task and any alternative paths that may exist for the same task.

In addition to eliminating the need to define the navigation path(s) for every task, LeNDI supplies the necessary input to the task modeling process (T3) of Figure 3.1, done by Mathaino. In the earlier versions of LeNDI and before developing the interaction-pattern discovery process, it was required to collect task specific traces to use as input for T3. Each set of such traces are multiple executions of the same task with different parameters that cover all the navigational and input and output possibilities of the same task, without any mistakes or spurious navigation. Then automatically, Mathaino analyzes the user inputs on the snapshots of the instances of each task to classify them to constants, derived variables, redundant values, range variables or unpredictable variables. Additionally, the Mathaino engineer manually highlights on the snapshots of each task instance the areas that contain the displayed information required to successfully complete the task. These highlighted locations are analyzed to infer the fixed or relative locations of the outputs of interest.

By introducing the process and algorithms of interaction pattern mining, LeNDI eliminated the need to collect multiple task specific traces. The regular interaction traces collected to build the state-transition model of legacy CUI are also used to discover interaction patterns, retrieve the instance of these patterns and feed them to Mathaino as multiple executions of the same task.

Interaction pattern discovery is a three-step process. It starts by some necessary preprocessing that transforms the data to the format needed by the mining algorithm and also reduces its size. Then, the mining algorithm is applied to discover the patterns that meet a user-defined interestingness criterion. The user usually reviews the discovered patterns and changes the interestingness criterion to narrow or widen the results set as needed or to see the effect of changing some parameters in the criterion on the results. Finally, s/he analyzes and comprehends the discovered patterns to distinguish the useful patterns from spurious navigational segments.

It is often the case that there are alternative paths to accomplish the same user task, e.g., the user can enter some value directly or open a list of choices to choose from. Additionally, it is possible to invoke some screens irrelevant to the user task intentionally or unintentionally, e.g., help screens or error messages. Due to these factors, it is important that the algorithm used for interaction pattern mining accommodates a user-defined level of noise in the instances of the patterns retrieved. This is done by defining the maximum number of "insertion errors" allowed in any instance of a pattern in order for it to be counted or considered. In this context, insertion errors are extra snapshots that may exist in the instances of a pattern, due to user mistakes or due to the existence of alternative paths for the same task. The type of patterns retrieved are called approximate patterns with insertion errors.

In [ESS02a], we introduced a more restricted version of this problem, in which exact interaction patterns with no insertion errors are discovered using an Apriori-based algorithm. But, since this limits the number and type of patterns retrieved, LeNDI needed to accommodate insertion errors. Since the existing sequential pattern mining algorithms did not address exactly the problem we have in hand, there was a need to develop a tailored algorithm to handle this problem. Thus we developed two algorithms for approximate interaction pattern mining with insertion errors, which are Interaction Pattern Miner (IPM), a breadth first algorithm, and Interaction Pattern Miner 2 (IPM2), a depth first algorithm. IPM requires more memory than IPM2 but is faster. This gives LeNDI analyst a choice between speed and memory usage depending on the trace set analyzed.

This chapter introduces the concept of interaction patterns and their use within CelLEST project and their other potential uses. Then, it introduces the problem of interaction pattern mining in traces of interaction with legacy CUIs and two novel algorithms for solving it, IPM and IPM2. The rest of this chapter is organized as follows. First, section 6.1 provides an example interaction pattern to illustrate how such patterns look like and how they can be represented. Section 6.2 formulates the interaction pattern mining problem. Section 6.3 describes the necessary simple preprocessing that is performed on the interaction traces before pattern discovery. Sections 6.4 and 6.5 describe the two algorithms developed for interaction pattern mining, IPM and IPM2. Section 6.6 is a brief on the post-discovery analysis of interaction patterns. Section 6.7 presents a case study and an evaluation and comparison of IPM and IPM2. Section 6.8

includes some final comments and discussion of other possible representations and uses of interaction patterns.

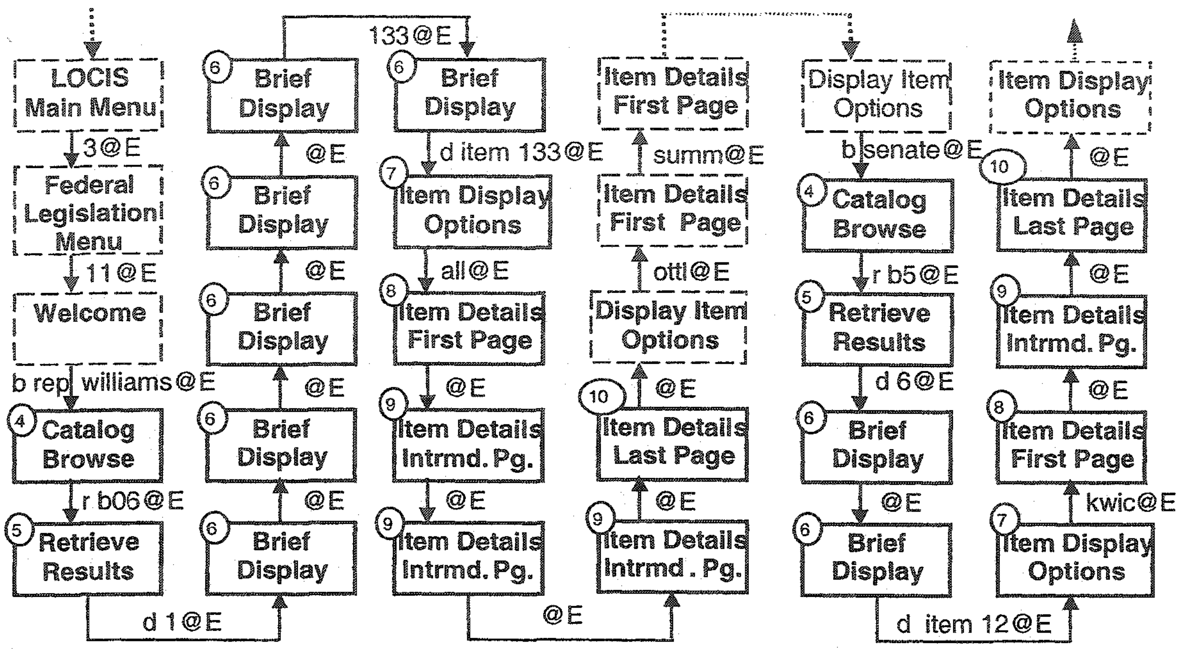
6.1 An Example Interaction Pattern

This section describes what an interaction pattern is and how it looks like and how it is represented. To do so, a user interacted with the Library of Congress Information System (LOCIS) [LOCIS] through its IBM 3270 public connection and performed a number of information retrieval tasks repeatedly, while LeNDI recorded this interaction. Figure 6.1(a) below shows a segment of the recorded interaction trace. Boxes represent screen snapshots and arrows represent transitions from one snapshot to another. The labels on the arrows are the user actions performed on the corresponding snapshots. The Ids in the circles at the upper left corners of the snapshots are the cluster Ids given to them by LeNDI after behavior modeling. Figure 6.1(b) shows the corresponding part of the state-transition model inferred by LeNDI from this trace. The boxes represent legacy screens or the nodes of the model. The numbers in the corners of the screens are the Ids given to them by LeNDI. Associated with each Id is the predicate or signature of the corresponding screen. The arrows are the model edges and the labels on them are the user action models.

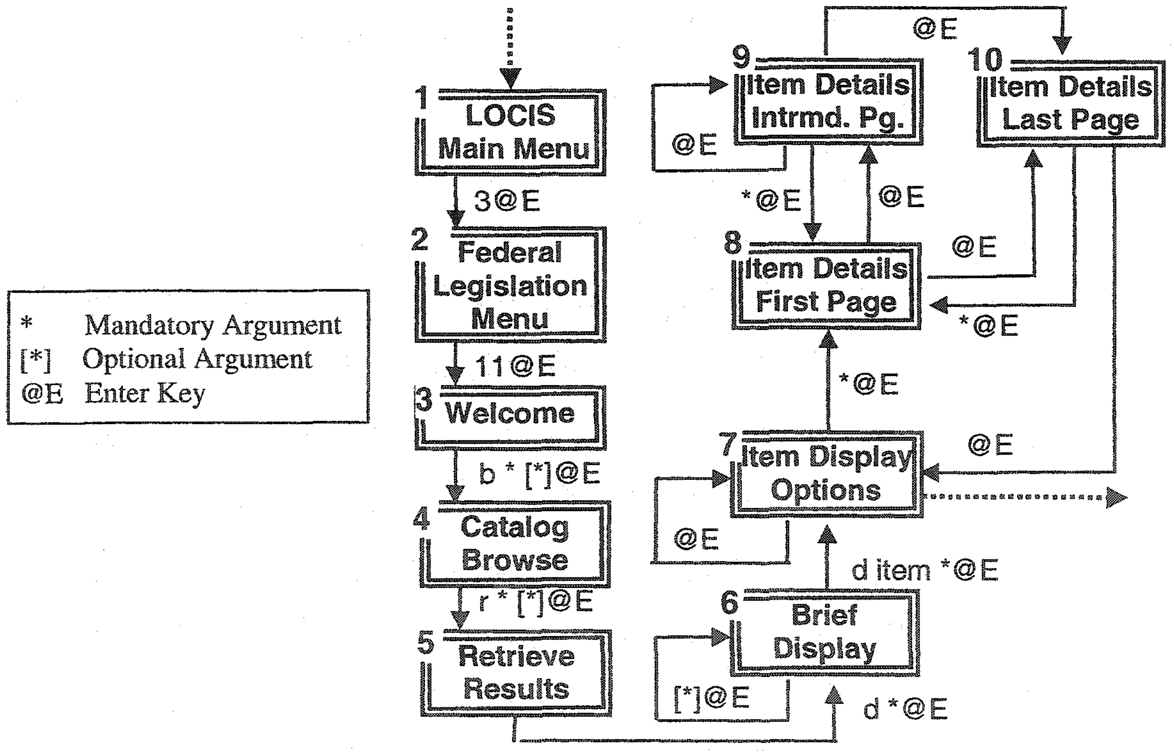
The portion of the trace shown in Figure 6.1(a) starts by the user making the menu selections needed to open the relevant library catalog. Then, the trace shows two very similar segments of navigating LOCIS in solid line boxes that occurred apart from each other in the trace. They represent two different executions of the same user task. In this task, the user issued a *browse* (*b*) command with some keyword(s) to browse the relevant part of the library catalog file. Then he issued a *retrieve* (*r*) command to retrieve a subset of the catalog items. Then, he displayed brief information about the items in this set using *display* (*d*) command. Finally, he selected an item using the *display item* (*d item*) command to display its full or partial information, e.g., the full legislation, its abstract, its list of sponsors, its official title, etc.

If a sufficient number of instances of this user task appear in the recorded traces and meet some user-defined criterion for pattern interestingness, LeNDI can discover that these instances represent a candidate interaction pattern, even if some of them include some insertion errors. Figure 6.2(a) shows the two similar navigation segments of Figure

6.1(a). One can see that the user accessed the same screens, in both segments, but he accessed a different number of snapshots of screens 6 and 9. The pattern corresponding to these navigation segments or task instances is $\{4^+,5,6^+,7^+,8^+,9\}$, where '+' means one or more instances of the preceding screen Id. A '+' is added after screen Ids 4 and 7 in this pattern because other instances of the pattern had multiple consecutive occurrences of these Ids. Sections 6.4 and 6.5 describe how this pattern and other ones are discovered. Figure 6.2(b) shows a diagrammatic representation of the discovered pattern, augmented with extra semantic information. Note that a mixture of constant and unpredictable values needs to be provided as input to perform this task. Some of the unpredictable variables are mandatory, represented by '*', and others are optional, represented by [*]. Other semantic information, like the outputs of interest to the user on each screen, needs to be added to the pattern before turning it into a complete task model.

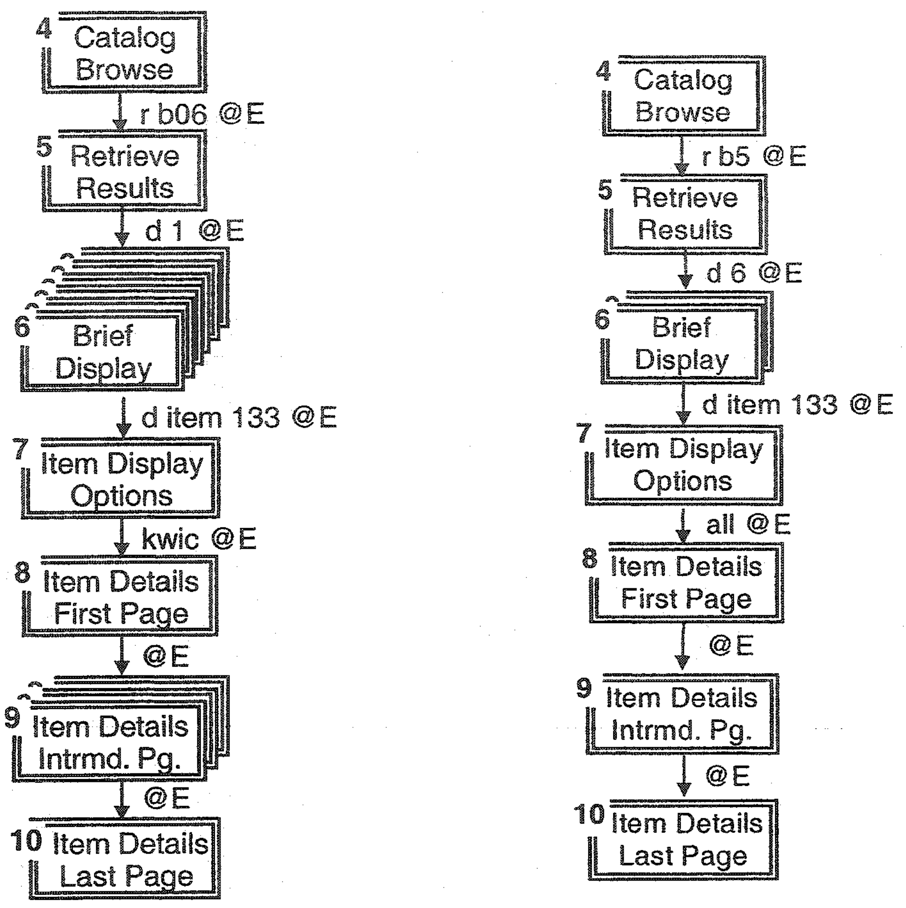


(a) A segment of an interaction trace with LOCIS.

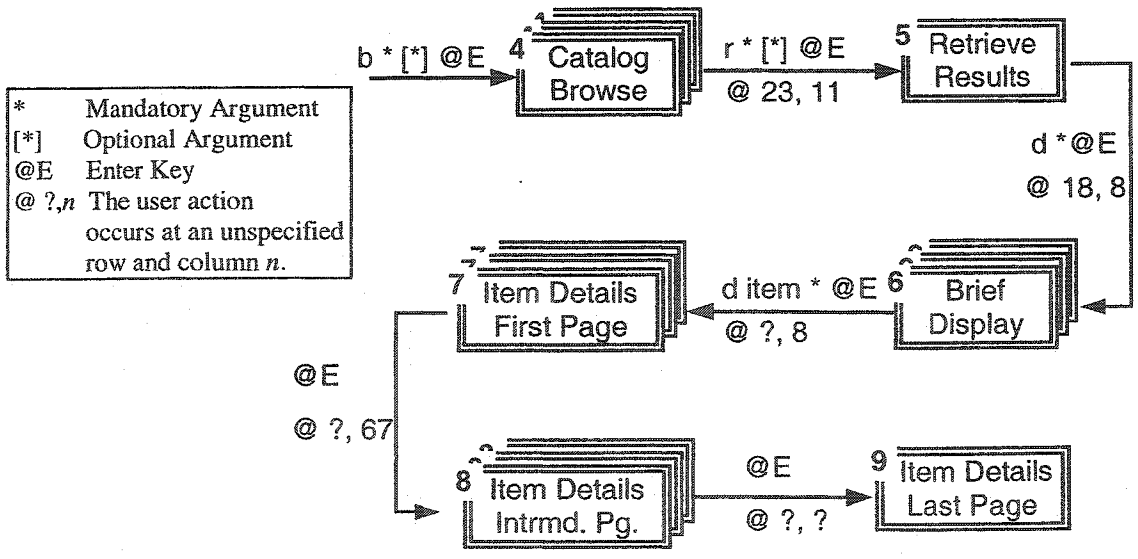


(b) The state-transition graph part corresponding to the segment in (a).

Figure 6.1. An Example Trace of User Interaction with the Library of Congress Information System (LOCIS) with Multiple Executions of the Same Task.



(a) Similar Navigation Segments of LOCIS That Represent The Same User Task.



(b)The corresponding interaction pattern.

Figure 6.2. Similar Navigation Subsequences of The LOCIS Trace of Figure 6.1(a) and The Corresponding Interaction Pattern Augmented with Action Locations.

6.2 Problem Formulation

This section provides the terminology and formulation of the problem of interaction pattern mining in the recorded traces of interaction with a legacy user interface.

1. Let A be the ***alphabet*** of legacy screen Ids, i.e., the set of Ids given by LeNDI to the screens of the legacy system under analysis.
2. Let $S = \{s_1, s_2, \ldots, s_n\}$ be a set of sequences. Each ***sequence*** s_i is an ordered set of screen Ids from A that represents a recorded trace of interaction between the user interface of the legacy system and one of its users, similar to the partial trace shown in Figure 6.1(a).
3. An ***episode*** e, is an ordered set of screen Ids occurring together in a given sequence.
4. A ***pattern*** p is an ordered set of screen Ids that exists in every episode $e \in E$, where E is a set of episodes of interest according to some user-defined criterion c. E and e are said to "support" p. The individual Ids in an episode e or a pattern p are referred to using square brackets, e.g., $e[1]$ is the first Id of e. Also, $|e|$ and $|p|$ are the number of items in e and p respectively.
5. If a set of episodes E supports a pattern p, then the first and last Ids in p must be the first and last Ids of any episode $e \in E$, respectively, and all Ids in p should exist in the same order in e, but e may contain extra Ids, i.e., $|p| \leq |e| \; \forall \; e \in E$. Formally,
   - $p[1] = e[1] \qquad \forall \; e \in E$,
   - $p[|p|] = e[|e|] \qquad \forall \; e \in E$, and
   - $\forall$ pair of positive integers (i, j), where $i \leq |p|$, $j \leq |p|$ and $i < j$, $\exists \; e[k] = p[i]$ and $e[l] = p[j]$ such that $k < l$.

   The above predicate defines the class of patterns that we are interested in, namely, ***approximate interaction patterns*** with at most a predefined number of insertions. For example, the episodes {2,***4***,3,4}, {2,***4***,3,***2***,4} and {2,3,4} support the pattern {2,3,4} with at most 2 insertions per episode, which are shown in bold italic font.
6. An ***exact interaction pattern*** q is a pattern supported by a set of episodes E such that none of its instances has insertion errors
   - $q[i] = e[i] \qquad \forall \; e \in E$ and $1 \leq i \leq |q|$

7. The ***location list*** of a pattern p, written as *loclist* (p), is a list of triplets (*seqnum*, *startLoc*, *endLoc*), each is the location of an episode $e \in E$, where s_{seqnum} is the Id of the sequence containing e. *startLoc* and *endLoc* are the locations of $e[1]$ and $e[|e|]$ in s_{seqnum}, respectively.
8. The ***support*** of a pattern p, written as *support* (p), is the number of episodes in S that support p. Note that *support* (p) equals the length of *loclist* (p), i.e., *loclist* (p).length.
9. The ***density*** of a pattern p, supported by a set of episodes E, is written as *density* (p) and is defined as the ratio of $|p|$ to the average episode length of episodes $\in E$:

   $$density\ (p) = \frac{|p| * support\ (p)}{\sum_{e \in E} |e|}$$

10. A ***qualification criterion*** c, or simply ***criterion***, is a user defined quadruplet (*minLen*, *minSupp*, *maxError*, *minScore*). Given a pattern p, the ***minimum length*** *minLen* is a threshold for $|p|$. The ***minimum support*** *minSupp* is a threshold for *support* (p). The ***maximum error*** *maxError* is the maximum number of insertion errors allowed in any episode $e \in E$. This implies that $|e| \leq |p| + maxError\ \forall\ e \in E$. The ***minimum score*** *minScore* is a threshold for the scoring function used to rank the discovered patterns. This function is:

    $$score\ (p) = \log_2 |p| * \log_2 support(p) * density(p)$$

    Experiments showed that this function is suitable and sufficient for the application in hand as it considers and balances between the pattern length, its support and its density. The default values for *minLen*, *minSupp*, *maxError* and *minScore* are 2, 2, 0 and 0 respectively. Other scoring functions can be used depending on the application.
11. A ***maximal pattern*** is a pattern that is not a sub-pattern of any other pattern with the same support.
12. A ***qualified pattern*** is a pattern that meets the user-defined criterion, c.
13. A ***candidate pattern*** is a pattern under analysis that meets the *minSupp* and *maxError* conditions.

Given the above definitions, the problem of interaction pattern discovery can be formulated as follows:

Given:

(a) an alphabet *A*,

(b) a set of sequences *S*, and

(c) a user criterion *c*

Find all the qualified maximal patterns in *S*.

6.3 Preprocessing Interaction Traces

An interaction trace is initially represented as a sequence *s* of integer screen Ids. We denote this representation as *R0*. *R0* often contains repetitions, resulting from accessing many instances of the same screen consecutively, e.g., browsing many pages of a library catalog. Repetitions may result in missing some important patterns. For example, the two instances of the interaction pattern of Figure 6.2(b), shown in Figure 6.2(a), are {4,5,6,6,6,6,6,6,7,8,9,9,9,10} and {4,5,6,6,7,8,9,10}. The user may keep flipping the pages of the result set that resulted from querying the library catalog until reaching the needed items. Hence, a variable number of snapshots of screen 6 may exist in a task instance. The same applies to screen 9. Unless LeNDI can tolerate this type of variability during its pattern mining process, it would miss some of the instances of such pattern and possibly not discover this pattern altogether. To avoid this problem, LeNDI encode *s* using the run-length encoding algorithm [Way99] that replaces immediate repetitions with a count followed by the repeated Id. Repetition counts are stored separate from the sequence. This representation is called *R1*. Figure 6.3 shows *R0* and *R1* representations of the trace segment of one of the pattern instances of Figure 6.2(a).

R0 **: {4, 5, 6, 6, 6, 6, 6, 6, 7, 8, 9, 9, 9, 10,}**
R1 **: {4, 5, (6)6, 7, 8, (3)9, 10}**

Figure 6.3. Preprocessing Interaction Traces.

6.4 IPM: Breadth-first Discovery of Approximate Interaction Patterns

Interaction Pattern Miner (IPM) [ESS02b] is one of two algorithms developed and implemented in LeNDI to discover interaction patterns. IPM utilizes a common idea in the field of data mining (DM). The idea is to construct shorter candidate patterns that meet the user required minimum support (number of occurrences) and maximum number of insertion errors and then glue them together to construct longer candidate patterns. Every pattern constructed is examined to ensure that it still meets these two conditions, before it is used to construct longer patterns. If a constructed pattern does not have enough support, then it is discarded and not used for constructing longer patterns. IPM is a breadth-first algorithm because it generates all candidate patterns of length l before generating any candidate pattern of length l+1, and so on and so forth. This requires saving the location lists of all candidate patterns of length l, to use them to generate the location lists of the patterns of length l+1.

The input to IPM is a set of sequences S and a criterion c. IPM outputs all the qualified patterns in S. IPM consists of two distinct phases. First, it exhaustively searches the input sequences to identify all the candidate patterns of length 2 during an initialization phase (Algorithm 6.1a). For every such pattern, a location list is constructed. The candidate patterns are stored in a matrix $|A| \times |A|$ of pattern lists, *ptList*, whose rows and columns are labeled after the Ids $\in A$. Each cell *ptList*[i,j] of the matrix contains every pattern p, such that $p[2]= i$ and $p[|p|]= j$. For example, the pattern {1,3,4,2} is stored in *ptList*[3,2].

In the second phase, Algorithm 6.1b recursively extends the candidate pattern set. For every pair of patterns *p1* and *p2* of length l, if *prefix* (*p1*) = *suffix* (*p2*), a new pattern *p3* of length l+1 is generated, such that *p3* = *p2* + *p1*[l], and is then stored in *ptList*. *p1* can only extend patterns in *ptList* [*p1*[1], *p1*[l-1]]. For example, if *p1* = {1,3,4,2}, then it will be used to extend the patterns of length 4 in *ptList* [1, 4], which have the format {?,1,?,4}, where ? refers to any Id $\in A$. The extension will succeed only with patterns with matching suffixes, i.e., of the format {?,1,3,4}. The location list of the extended pattern *p3* is constructed from the location lists of *p1* and *p2* (Algorithm 6.1c). Locations of the episodes that support *p3* but have more than *maxError* insertion errors are excluded. If *support* (*p3*) $\geq$ *minSupp*, then *p3* and *loclist* (*p3*) are stored in *ptList*, otherwise *p3* is discarded. If *support* (*p3*) = *support* (*p1*) and/or *support* (*p3*) = *support* (*p2*), then *p1* and/or *p2* is marked as non-maximal. When no more candidates can be generated, the algorithm reports the qualified maximal patterns in *ptList*. The following is a step by step description of Algorithms 6.1a, 6.1b and 6.1c.

6.4.1 IPM Phase 1: Producing The Initial Candidate Pattern Set

Algorithm 6.1a implements the initialization phase of IPM. Step 1 creates the pattern list matrix, *ptList*. Step 2 is repeated for every input sequence $s_k \in S$. Step 2.a iterates over the Ids of s_k, from s_k [1] to s_k [$|s_k|$ - *maxError*-1]. For each Id, it iterates in the inner loop over its consecutive Ids up to *maxError*+1. Step 2.a.I uses each of these consecutive Ids to build a new pattern with original Id. For example if s_k = {1,3,2,3,4} and *maxError*=2, then s_k [1] will be glued to each of s_k[2], s_k[3] and s_k[4] separately, resulting in the generation of the new patterns {1,3}, {1,2} and {1,3}. Step 2.a.II adds the new pattern in *ptList,* if it is not already there. The location of the episode supporting the pattern is added to its location list in step 2.a.III. Step 2.b performs the same function as steps 2.a, but it handles the last *maxError* Ids of s_k. Note that the only cells of *ptList*, used by Algorithm 6.1a, are the diagonal cells. This is because for a pattern of length 2, *p*[2] is *p*[|*p*|], i.e., it is stored in *ptLis1* [*p*[2], *p*[|*p*|]], which is *ptLis1* [*p*[2], *p*[2]]. Step 3 removes from *ptList* any pattern whose support is less than *minSupp*.

```
Algorithm 6.1a: IPM Initial Phase
Input: An alphabet $A$, a criterion $c$ and a set of sequences $S$.
Output: All candidate patterns of length 2.
Steps:
1. Create a matrix $|A|$ x $|A|$ of pattern lists, ptList
2. For every trace $s_k \in S$, $1 \leq k \leq |S|$
   a. For $i = 1$ to $|s_k|$ - $maxError - 1$
      • For $j = i + 1$ to $i + maxError + 1$
        I.   Construct new pattern $p = \{s_k[i], s_k[j]\}$
        II.  If $p$ NOT in ptList [$s_k[j]$, $s_k[j]$] then Add $p$ to ptList [$s_k[j]$, $s_k[j]$ ]
        III. Add $(k,i,j)$ to ptList [$s_k[j]$, $s_k[j]$].getLocationList ($p$)
   b. For $i = |s_k|$ - $maxError$ to $|s_k|$ -1
      • For $j = i + 1$ to $|s_k|$
        I.   Construct new pattern $p = \{s_k[i], s_k[j]\}$
        II.  If $p$ NOT in ptList [$s_k[j]$, $s_k[j]$] then Add $p$ to ptList [$s_k[j]$, $s_k[j]$ ]
        III. Add $(k,i,j)$ to ptList [$s_k[j]$, $s_k[j]$].getLocationList ($p$)
3. For every $id \in A$
   a. For every pattern $p$ in ptList [id, id]
      • If support (ptList [id, id]) < minSupp then Remove $p$ from ptList [id, id]
```

Algorithm 6.1a. IPM Initial Phase.

6.4.2 IPM Phase 2: Generating Longer Candidate Patterns from Shorter Ones

Algorithm 6.1b implements the second phase of IPM. Step 2 iterates as long as more candidate patterns can be generated as indicated by the *morePatterns* flag, which is set to false in step 2.a, at the beginning of every new iteration. Step 2.b loops over every cell in *ptList* matrix. For every cell it access every pattern *p1* of length *l* and checks if *p1* can be used to extend any pattern *p2* from its end. Only the patterns in *ptList* [*p1*[1], *p1*[*l*-1]] are inspected because these are the ones whose second Id, *p2*[2] and last Id, *p2*[*l*] match *p1*[1] and *p1*[*l*-1], respectively. If extension is possible, i.e., *suffix* (*p2*) equals *prefix* (*p1*), then step 2.a.I generates the new pattern *p3* and step 2.a.II constructs its location list. Step 2.a.III checks if *p3* satisfies the minimum support condition. If yes, it adds *p3* to *ptList*, marks *p1* and/or *p2* as non-maximal if they have the same support as *p3* and sets the flag *morePatterns* to true to execute a new iteration. Step 2.c increments the pattern length counter *l* for the next iteration. When no more candidates can be generated, step 3 iterates over every cell in *ptList* and step 3.a access every candidate pattern in the cell. Step 3.a.I reports the pattern only if it is qualified and maximal.

Algorithm 6.1b: IPM Phase 2: Generating Long Candidate Patterns from Short Ones.

```
Input: A matrix of pattern lists, $ptList$, initialized with all candidate patterns of length 2
        and their location lists and a criterion $c$.
Output: All the qualified maximal patterns according to $c$.
Steps:
1. $l = 2$
2. Repeat
   a. $morePatterns$ = false
   b. For every $a \in A$ For every $b \in A$
      • For every pattern $p1$ in $ptList\ [a, b]$ with $|p1| == l$
         ❑ For every pattern $p2$ in $ptList\ [p1[1], p1[l\text{-}1]]$ with $|p2| == l$
            • If $suffix\ (p2) == prefix\ (p1)$ then
            I.   Construct new pattern $p3 = p2 + p1\ [l]$
            II.  Construct $loclist\ (p3)$  (Algorithm 6.1c)
            III. If $support\ (p3) \geq minSupp$ then
                 • Add $p3$ to $ptList\ [p1[1], p1[l]]$
                 • If $support\ (p3) == support\ (p1)$ then mark $p1$ as non-maximal
                 • If $support\ (p3) == support\ (p2)$ then mark $p2$ as non-maximal
                 • $morePatterns$ = true
   c. $l$++
   While $morePatterns$ == true
3. For every $a \in A$ For every $b \in A$
   a. For every pattern $p$ in $ptList\ [a, b]$
      I. If $|p| \geq minLen$ AND $score\ (p) \geq minScore$ AND $p$ is maximal then report $p$
```

Algorithm 6.1b. IPM Phase 2: Generating Long Candidate Patterns from Short Ones.

Algorithm 6.1c creates the location list of a new candidate pattern. It takes as input the location lists of two patterns *p1* and *p2* of length *l*, sorted by *seqnum* and *startLoc*. It outputs the location list of *p3*, where $p3 = p2 + p1\ [l]$. Step 2 iterates over the locations of the episodes supporting *p2*. Steps 2.a to 2.c retrieve *startLoc* and *endLoc* of such an episode *e2*. Step 2.d retrieves the locations of the episodes that support *p1* and satisfy some conditions. For such an episode *e1*:

- *e1* and *e2* should be in the same sequence
- *e1* should not be a sub-episode of *e2* and vise versa.
- The overlap of *e1* and *e2* should be at least *l*-1 long.
- The distance from *startLoc* of *e2* to *endLoc* of *e1*, inclusive, should be no more than *l* + 1 + *maxError*.

```
Algorithm 6.1c: Generating The Location List of a Candidate Pattern for IPM
Input: The location lists of patterns $p1$ and $p2$ of length $l$ and $maxError$.
Output: The location list of $p3$, where $p3 = p2 + p1\,[l]$.
Steps:
1. Create a empty location list $Loc3$
2. For $i = 1$ to $loclist\,(p2)$.length
   a. $loc2 = loclist\,(p2)$.getLocation($i$)
   b. $st = loc2.startLoc$
   c. $end = loc2.endLoc$
   d. Find a set $Loc1$ = (any $loc1 \in loclist\,(p1)$ such that $loc1.seqnum = loc2.seqnum$
                            AND $st < loc1.startLoc \le end - l + 1$
                            AND $end < loc1.endLoc \le st + maxError + l$ )
   e. For every $loc1 \in Loc1$
      • Add a triplet ($loc1.seqnum$, $st$, $loc1.endLoc$) to $Loc3$
3. Remove any duplicates from $Loc3$
4. Return $Loc3$
```

Algorithm 6.1c. Generating The Location List of a Candidate Pattern for IPM.

Step 2.e constructs the location list of *p3*. Step 3 removes duplicates from the list. Finally, step 4 reports the results back.

6.4.3 An IPM Application Example

This subsection illustrates the operation of IPM algorithm with a simple example. Assume:

(a) $A = \{1,2,3,4\}$,

(b) $S = \{\{1,3,2,3,4,3\},\{2,3,2,4,1,3\}\}$, and

(c) $c = (minLen, minSupp, maxError, minScore) = (2,2,1,0)$

Discover all the qualified maximal patterns in *S*.

Tables 6.1 to 6.3 show the steps of applying IPM. Patterns are enclosed between curved brackets, e.g., {2,1}, and their locations in the input sequences are between parentheses, e.g., (2,3,5). Candidate patterns are shown in bold. Patterns with insufficient support are shown in normal font for clarification, although they are not stored in *ptList* according to Algorithms 6.1a and 6.1b. Candidate patterns of the previous iteration that turned out to be non-maximal in the current iteration are shown in normal font and followed by ¬max.

Last Id 2nd Id	1	2	3	4
1	{2,1} (2,3,5) {4,1} (2,4,5)			
2		{1,2} (1,1,3) **{3,2} (1,2,3)(2,2,3)** {2,2} (2,1,3)		
3			**{1,3} (1,1,2) (2,5,6)** **{3,3} (1,2,4) (1,4,6)** **{2,3} (1,3,4) (2,1,2)** **{4,3} (1,5,6) (2,4,6)**	
4				**{2,4} (1,3,5)(2,3,4)** **{3,4} (1,4,5)(2,2,4)**

Table 6.1. The Matrix *ptList* after IPM Phase 1 (Algorithm 6.1a) for The Example of Subsection 6.4.3.

Last Id 2nd Id	1	2	3	4
1				
2		{3,2} ¬max	{3,2,3} (1,2,4)	**{3,2,4} (1,2,5)(2,2,4)**

3		{1,3,2}(1,1,3)
		{3,3,2}
		{2,3,2}(2,1,3)
		{4,3,2}
			{1,3} (1,1,2)(2,5,6)
			{3,3} (1,2,4)(1,4,6)
			{2,3} ¬max
			{4,3} ¬max
			{1,3,3} (1,1,4)
			{2,3,3} (1,3,6)
			{3,3,3}
			{4,3,3}
				{1,3,4}
				{3,3,4}(1,2,5)
				{2,3,4} (1,3,5)(2,1,4)
				{4,3,4}

4			{3,4,3} (1,4,6) **{2,4,3} (1,3,6)(2,3,6)**	{2,4} ¬max {3,4} ¬max

Table 6.2. The Matrix *ptList* after Iteration 1 of IPM Phase 2 (Algorithm 6.1b) for IPM Application Example of Subsection 6.4.3

Table 6.1 shows the pattern list matrix, *ptList*, containing the initial candidate patterns of length 2 generated by Algorithm 6.1a. Table 6.2 shows *ptList* after the first iteration of Algorithm 6.1b, during which all candidate patterns of length 3 were generated and non-maximal patterns of length 2 were marked. Table 6.3 shows *ptList* after the second iteration of Algorithm 6.1b. Table 6.4 shows the discovered qualified patterns, their support, density and score.

Last Id / 2nd Id	1	2	3	4
1				
2			{3,2,4,3}(1,2,6)(2,2,6)	{3,2,4}¬max
3			{1,3} (1,1,2)(2,5,6) {3,3} (1,2,4)(1,4,6)	{2,3,4} (1,3,5)(2,1,4)
4			{2,4,3}¬max	

Table 6.3. The Matrix *ptList* after Iteration 2 of IPM Phase 2 (Algorithm 6.1b) for IPM Application Example of Subsection 6.4.3.

Pattern p	$\|p\|$	*Support* (p)	*Density*(p)	*Score* (p)
{3,2,4,3}	4	2	0.80	1.60
{2,3,4}	3	2	0.86	1.36
{1,3}	2	2	1.00	1.00
{3,3}	2	2	0.67	0.67

Table 6.4. All The Maximal Qualified Patterns in *S* for IPM Application Example of Subsection 6.4.3

6.5 IPM2: Depth-first Discovery of Approximate Interaction Patterns

Interaction Pattern Miner 2 (IPM2) [ESS02c] is LeNDI's second interaction pattern mining algorithm. Like IPM, IPM2 develops longer candidate patterns from shorter ones. Unlike the breadth-first strategy of IPM, IPM2 uses a depth-first strategy. It requires less memory than IPM but is slower. Hence, it can handle bigger data sets than IPM. IPM2 was developed to offer the LeNDI analyst a choice. If s/he is analyzing a small trace set, then IPM is faster. If the data set is too big for IPM, then IPM2 can analyze it.

IPM2 extends a pattern of length 2 with another pattern of length 2 to form a pattern of length 3. If the latter has enough support, then it is extended again with another pattern of length 2 and so on and so forth until no further extension is possible. Then, IPM2 backtracks, reports any maximal qualified pattern found and continues depth-first extensions. This eliminates the need to store all the patterns of length l at the same time in a matrix $|A|$ x $|A|$ of patterns and their location lists, which can be memory exhaustive if the size of the data and alphabet A is big. So, IPM2 is more suitable for big systems with numerous screens. This advantage comes at the cost of generating more candidate patterns than IPM, and hence, more computational time. An evaluation and comparison of both algorithms is provided in section 6.7.

IPM2 consists of two distinct phases. Phase 1 exhaustively searches the input sequences to find all the candidate patterns of length 2 that meet the “minimum support” and “maximum error” conditions (Algorithm 6.2a). For every such pattern, a location list is constructed. The patterns are stored in a vector of length $|A|$ of pattern lists, *ptListVec*, whose cells are labeled after the Ids of *A*. Each cell *ptListVec*[*i*] contains all patterns *p*, such that *p*[1]= *i*. For example, the pattern {1,3} is stored in *ptLisVect*[1].

Phase 2 (Algorithm 6.2b) recursively extends each candidate pattern in *ptListVec* using a depth-first approach. If an extension of a candidate pattern *p1* using another pattern *p2* produces a new candidate pattern *p3* = *p1*+*p2*[2], then *p3* is extended further. *p1* can be extended only with patterns in *ptListVec* [*p1*[|*p1*|]], i.e. patterns of length 2 whose first Id is the same as the last Id of *p1*. The location lists of *p1* and *p2* are used to construct that of *p3* (Algorithm 6.2c). The locations of the episodes that support *p3*, but have more insertion errors than *maxError* are excluded. If *support* (*p3*) $\geq$ *minSupp* then *p3* is extended further using the patterns in *ptListVec* [*p3*[|*p3*|]], otherwise *p3* is ignored and the algorithm records *p1* if it is qualified and then backtracks. During backtracking and after reporting a pattern *p1*, the algorithm examines the parent pattern *p0* of *p1*. Since *p0* is a sub-pattern of *p1*, it is a candidate pattern also. If *p0* is qualified and *support* (*p0*) > *support* (*p1*), i.e., it is not non-maximal relative to *p1*, then it is recorded too. After trying to extend all patterns in *ptListVec*, non-maximal patterns are removed and only qualified maximal patterns are reported.

6.5.1 IPM2 Phase 1: Producing the Initial Candidate Pattern Set

Algorithm 6.2a describes the first phase of IPM2. Step 1 creates a vector *ptListVec* of pattern lists. PatternList is a hash-table-like data structure that can hold a list of hashed patterns. Step 2 is repeated for every input sequence $s_k \in S$. Step 2.a iterates over the Ids of s_k, from s_k [1] to s_k [$|s_k|$ - *maxError* -1]. In step 2.a.I, each Id is used to build a pattern with each of its consecutive Ids up to *maxError* +1. For example if s_k = {1,3,2,3,4,3} and *maxError* = 2, the first Id will be tried with each of its next three resulting in the generation of these patterns {1,3}, {1,2} and {1,3}. A new pattern is stored in *ptListVec,* if it is not there already and the location of the episode supporting it is added to its location list. Steps 2.b does the same as step 2.a, but it handles the last *maxError* Ids of s_k. Step 3 removes any non-candidate pattern *p*, i.e. patterns with *support* (*p*) < *minSupp*.

```
Algorithm 6.2a: IPM2 Initial Phase
Input: An alphabet $A$, a criterion $c$ and a set of sequences $S$.
Output: All candidate patterns of length 2.
Steps:
1. PatternList $ptListVec$ [$|A|$]
2. For every trace $s_k \in S$, $1 \le k \le |S|$
   a. For $i = 1$ to $|s_k|$ - $maxError$ -1
      I. For $j = i$ +1 to $i$ + $maxError$ +1
         ❑ Construct new pattern $p = s_k[i] + s_k[j]$
         ❑ If $p$ NOT in $ptListVec$ [$s_k[i]$] then Add $p$ to $ptListVec$ [$s_k[i]$]
         ❑ Add $(k,i,j)$ to $loclist$ ($p$)
   b. For $i = |s_k|$ - $maxError$ to $|s_k|$ -1
      I. For $j = i$ +1 to $|s_k|$
         ❑ Construct new pattern $p = s_k[i] + s_k[j]$
         ❑ If $p$ NOT in $ptListVec$ [$s_k[i]$] then then Add $p$ to $ptListVec$ [$s_k[i]$]
         ❑ Add $(k,i,j)$ to $loclist$ ($p$)
3. For every $id \in A$
   a. For every pattern $p$ in $ptListVec$ [$id$]
      • If $loclist$ ($p$).length < $minSupp$ then Remove $p$ from $ptListVec$ [$id$]
```

Algorithm 6.2a. IPM2 Initial Phase.

6.5.2 IPM2 Phase 2: Generating Longer Candidate Patterns from Shorter Ones

Algorithm 6.2b generates longer patterns from shorter ones. Step 1 creates a pattern list, called *resultsIPM2* to store the discovered patterns. Step 2 iterates over every cell in *ptListVec* using the iterator *id* and for each cell *ptListVec* [*id*], it iterates over each pattern in it. For every such pattern *p*, step 2.a calls the procedure "Extend (*p1*)", which returns all the qualified extension patterns of *p* that are maximal relative to each other, i.e., none of them is a sub-pattern of another with the same support. Step 2.b adds the discovered extensions of *p* to *resultsIPM2*. Step 3 removes any non-maximal pattern from the final results. Step 4 reports the final results in *resultsIPM2*.

The "Extend (*p1*)" procedure works as follows. Step 1 creates a pattern list *extensionResults* to hold the patterns resulting from successful extensions of the parameter pattern *p1*. Step 2 iterates over every pattern *p2* that can extend *p1*, i.e., every pattern whose first Id is the same as the last Id of *p1*. Steps 2.a and 2.b construct the extended pattern *p3* and its location list. Step 2.c tests if the support of $p3 \ge minSupp$. Steps 2.c.I to 2.c.III are executed in case of True and step 2.d.I is executed in case of False.

Algorithm 6.2b: IPM2 Phase 2: Generating Long Candidate Patterns from Short Ones.

```
Input: A vector of pattern lists, ptListVec, initialized with all candidate patterns of length
       2 and their location lists and a criterion c.
Output: All the maximal qualified patterns according to c.
Steps:
1. Create new PatternList resultsIPM2
2. For every $id \in A$ For every pattern p in ptListVec [id]
   a. Create new PatternList tempResults = Extend (p)
   b. Merge tempResults with resultsIPM2
3. Remove non-maximal patterns from resultsIPM2
4. Report resultsIPM2

PatternList Extend (p1)
1. PatternList extensionResults
2. For every pattern p2 in ptListVec [p1[|p1|]]
   a. Construct new pattern p3 = p1 + p2 [|p2|]
   b. Construct the location list of p3 (Algorithm 6.2c)
   c. If support (p3) $\geq$ minSupp then
      I. Create new PatternList tempResults = Extend (p3)
      II. Merge tempResults with extensionResults
      III. If support (p1) > support (p3) then
         • If |p1| $\geq$ minLen AND score (p1) $\geq$ minScore then
            ❑ If p1 is NOT in extensionResults then add p1 to extensionResults
   d. Else
      I. If |p1| $\geq$ minLen AND score (p1) $\geq$ minScore then
         • If p1 is NOT in extensionResults then add p1 to extensionResult
3. Return extensionResults
```

Algorithm 6.2b. IPM2 Phase 2: Generating Long Candidate Patterns from Short Ones

In case of a successful extension, step 2.c.I extends the new candidate *p3* more by calling Extend (*p1*) with *p3* as a parameter. Step 2.c.II adds the qualified patterns resulting from extending *p3* to *extensionResults*. Step 2.c.III adds *p1* to *extensionResults* if it has more support than its successful extension *p3*, it is qualified and it is not already in *extensionResults*. In case of failing to extend *p1* using *p2*, then the extension pattern *p3* is ignored and steps 2.d.I adds *p1* to the results list *extensionResults* if it is qualified and it is not already in *extensionResults*. Step 3 reports all the qualified maximal (relative to one another) extension patterns of *p1*.

Algorithm 6.2c describes the process of creating the location list of a new candidate pattern. It combines the locations lists of two patterns *p1* and *p2*, where $|p2| = 2$, to provide

---

```
Algorithm 6.1c: Generating The Location List of a Candidate Pattern for IPM2
Input: The location lists of patterns p1 and p2 and maxError.
Output: The location list of p3, where p3 = p1 + p2 [2].
Steps:
1. Create an empty location list listLoc3
2. For i = 1 to loclist (p1).length
   a. loc1 = location i in loclist (p1)
   b. Find a set Loc1 = (any loc2 ∈ loclist (p2) such that
                  loc2.seqnum == loc1.seqnum AND
                  loc2.startLoc == loc1.endLoc AND
                  loc2.endLoc ≤ loc1.startLoc + maxError + |p1|)
   c. For every loc1 ∈ Loc1
      • Add (loc2.seqnum, start, loc2.endLoc) to listLoc3
3. Remove any duplicates from listLoc3
4. Return listLoc3
```

Algorithm 6.1c. Generating The Location List of a Candidate Pattern for IPM2

the location list of *p3*, where $p3 = p1 + p2$ [2]. The input locations lists are sorted by *seqnum* and *startLoc*. Step 2 iterates over the locations of the episodes supporting *p1*. Steps 2.a retrieves the location of such an episode *e1*. Step 2.b retrieves the locations of the episodes that support *p2* and satisfy these conditions, assuming such an episode *e2*:

- *e1* and *e2* should be in the same sequence
- *e1* should not be a sub-episode of *e2* and vise versa.
- *e1* and *e2* should overlap in exactly one location which is $e1[|e1|]$.
- The distance from *startLoc* of *e1* to *endLoc* of *e2*, inclusive, should be no more than $|p1| + 1 + maxError$.

Step 2.c constructs the location list of *p3* and step 3 removes any duplicates. Finally, step 4 reports the results back.

6.5.3 An IPM2 Application Example

This simple example illustrates the operation of IPM2. Assume:

(d) Let $A = \{1,2,3,4\}$,

(e) $S = \{\{1,3,2,3,4,3\},\{2,3,2,4,1,3\}\}$, and

(f) $c = (minLen, minSupp, maxError, minScore) = (3,2,1,0)$.

Discover all the qualified maximal patterns in *S* according to *c*.

First Id	1	2	3	4
Candidate Patterns (bold)	{1,2} (1,1,3) **{1,3} (1,1,2)(2,5,6)**	{2,1} (2,3,5) {2,2} (2,1,3) **{2,3} (1,3,4) (2,1,2)** **{2,4} (1,3,5) (2,3,4)**	**{3,2} (1,2,3) (2,2,3)** **{3,3} (1,2,4) (1,4,6)** **{3,4} (1,4,5) (2,2,4)**	**{4,3} (1,5,6)(2,4,6)** {4,1} (2,4,5)

Table 6.5. ***ptListVec*** **after IPM2 Initial Phase (Algorithm 6.2a) for IPM2 Application Example of Subsection 6.5.3.**

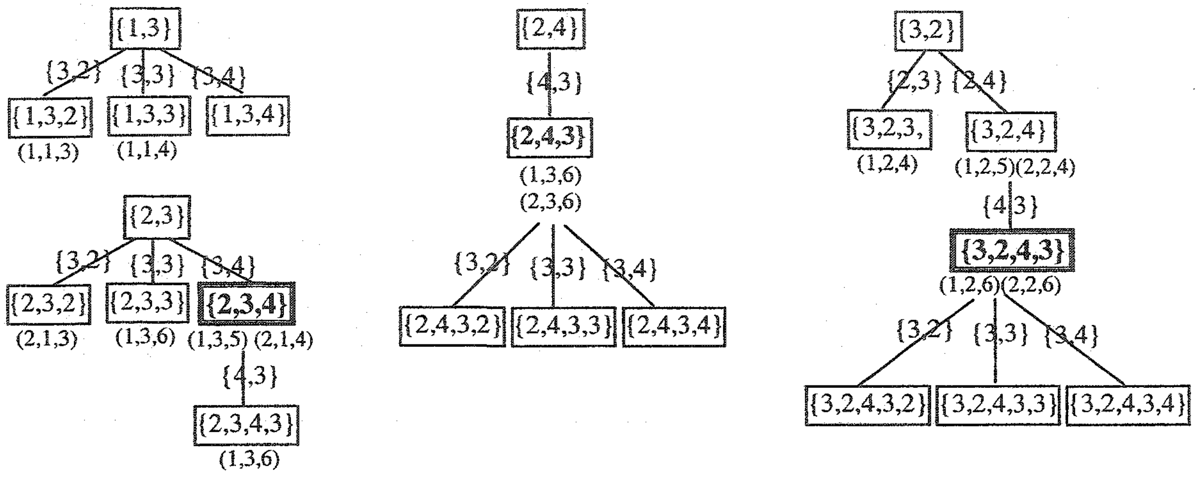


Figure 6.4. The Application of IPM2 Phase 2 (Algorithm 6.2b) for IPM2 Application Example of Subsection 6.5.3.

Table 6.5 shows the result of applying the initial phase of IPM2 to *S*. The second row corresponds to the cells of *ptListVec*. Patterns are enclosed between curved brackets, e.g., {1,2}, and their locations in the sequences are shown next to them between parentheses, e.g., (1,1,3). Candidate patterns are shown in bold. Patterns with insufficient support are shown in normal font. They are removed from *ptListVec* at the end of Algorithm 6.2a, but are kept in Table 6.5 for clarification. Figure 6.4 shows partial application of Algorithm 6.2b to extend 4 of the 8 candidate patterns in Table 6.5. The patterns in boxes are the ones being extended or resulting from extension. The patterns on the arcs are the ones used for extension. The location list of every generated pattern is shown under its box. Qualified patterns that are reported by the sub-procedure "Extend (*p1*)" are shown in bold font. Maximal qualified patterns, returned by IPM2, are in double-line boxes. Note that the pattern {3,2,4} is qualified but not reported by "Extend (*p1*)" because its extension {3,2,4,3} which has the same support, is reported first, while {2,4,3} is reported by "Extend (*p1*)" but is removed at the end of phase 2 for being non-maximal. Table 6.6 shows the discovered maximal qualified patterns, their support, density and score.

Pattern p	$\|p\|$	*support* (p)	*Density*(p)	*Score* (p)
{2,3,4}	3	2	0.86	1.36
{3,2,4,3}	4	2	0.80	1.60

Table 6.6. All The Maximal Qualified Patterns in *S* for IPM2 Application Example of Subsection 6.5.3.

6.6 Understanding The Extracted Patterns

After reviewing the discovered patterns, the criterion c can be modified to narrow or widen the results set, if too few or too many patterns were retrieved. Also, any group of patterns, whose score and/or support are within specific range(s), can be compacted by removing any pattern that is a sub-pattern of another pattern, even if it is maximal.

This interactive step of scoping out and "cleaning" the extracted interaction patterns is crucial in identifying the usage scenarios corresponding to the functional requirements of the legacy application. Methodologically, the longer the recorded traces and the "stricter" the criterion c, the more likely it becomes to discover true usage scenarios, since all "noise patterns" should not gain enough support when evaluated in the context of long-term use. However, the LeNDI analyst has to decide which of the discovered patterns correspond indeed to real usage scenarios. To do so, the analyst retrieves and reviews instances of the interaction patterns discovered. Then, s/he can exclude trivial patterns, accept complete real patterns and/or complete partial patterns.

6.7 Evaluation

To evaluate our interaction pattern mining process and algorithms, we applied them to traces of interaction with a number of legacy systems [ESS02c, SES02]. Additionally, we tested the scalability of the algorithms using very long traces generated artificially using a simulator. Finally, we applied our interaction pattern mining algorithms, IPM and IPM2, to a different domain. We used them to discover frequent user navigation patterns from server logs of a focused web site, i.e., one that is usually navigated in a systematic task-driven way in support of an ongoing process [ES03, NSE02]. The web site we used was a university course site, where new material and assignment are posted weekly, and students access them in a task-oriented way. The goal in that application was to recommend web pages to the users based on their navigation history if it matches the prefix(es) of some of the discovered pattern(s). In such case, the suffix(es) of the pattern(s) are recommended to the user.

This section reports two different evaluation experiments. The first is a case study, during which, traces of interaction between a library information system and a user were recorded and then mined to discover what frequent tasks the user was performing. This case study demonstrates the applicability and usefulness of the interaction pattern mining process and shows how much human input is required to recover accurate representations of the frequent user tasks. Second, a comparison between the memory requirements and speed of IPM and IPM2 is performed using long artificial traces that were generated using LeNDI's Legacy System Trace Generator (LSTG), as described in details shortly.

6.7.1 A Case Study of Interaction Pattern Mining in the Traces of LOCIS

This section presents a demonstrative case study of recovering the usage scenarios or interaction patterns from recorded traces of interaction with the Library of Congress Information System (LOCIS) [LOCIS], via its public 3270 connection. A user conducted five interaction sessions with LOCIS, during which, he repeatedly performed various information retrieval tasks about federal legislation. Each session was captured in an interaction trace. Thus, $S = \{s_1, s_2, s_3, s_4, s_5\}$, where $|s_1|$, $|s_2|$, $|s_3|$, $|s_4|$ and $|s_5|$ are 454, 185, 369, 410 and 239, respectively. In total, 1657 snapshots were captured in these traces. Part of s_1 is shown in Figure 6.1(a). LeNDI was used to build the state-transition model corresponding to S. Part of this model is shown in Figure 6.1(b). The model has 27 nodes. Each node corresponds to a LOCIS system screen. Thus, $A = \{1,2,3,\ldots,27\}$. The screen descriptions are provided in Table 6.7. The frequency (Fr.) of each screen is the number of times it was recorded in S.

After preprocessing S, IPM2 was applied to S several times to discover the user's interaction patterns with LOCIS, and model them. Several runs were done with different parameters for the criterion c (*minLen*, *minSupp, maxError*, *minScore*) to see the effect of changes in c. The results of the most interesting runs are recorded in Tables 6.8 to 6.11.

6.7.1.1 The First Run

The first run used c = (6, 9, 0, 7). Its results are shown in Table 6.8 ordered by their score. Six patterns were discovered and the results were further compacted by removing the patterns that are subsets of other patterns. The removed patterns are shown in gray. Then, sample instances of each interaction pattern were reviewed to see how well it corresponds to a real user task, i.e., to a usage scenario of the system. This inspection revealed that the patterns in bold, 2, 3 and 4, closely correspond partially or fully to three repetitive user tasks. The actual interaction patterns of the three tasks discovered are:

1. 4^+-5-6^+-7^+-8^+-9^+-10
2. 4^+-14-15^+-6^+-7^+-8^+
3. 22-23-22-6^+-7^+-8^+-9^+-10

Note that *S* is in *R1* format. By checking the instances of each pattern in the original traces in *R0* format, we saw which screens are consecutively repeated and added to them '+' signs. A Complete description of the tasks of these interaction patterns follows shortly.

Id	Screen Description	Fr.
1	Main LOCIS Menu	18
2	Federal Leg. Menu	13
3	Welcome	13
4	Browse Result	132
5	Retrieve Result	55
6	Brief Display	268
7	Display Item Options	201
8	Display item 1/1 or 1st	161
9	Display item (2/n or more/n) page	178
10	Display item (n/n) page	81
11	Error	91
12	Search History	62
13	Display List	5
14	Select Result[12]	33

Id	Screen Description	Fr.
15	Combine Result[8]	37
16	Release Result[9]	9
17	Comments & Logoff	3
18	Goodbye	6
19	Ready for a Command	3
20	System Message	43
21	Livt Results (1/1) page[10]	44
22	Expand Results (1/n)[11]	63
23	Expand/Livt Results (n/n, i.e. last) page	47
24	Expand/Livt Results (2/n or more/n) page	5
25	Livt Results (1/n) page	19
26	Expand Results (1/1)	23
27	Help	44

Table 6.7. LOCIS Screen Descriptions and Frequencies for The Interaction Pattern Mining Case Study of Subsection 6.7.1.

[8] *Combine* command creates a new set of records by logically combining previously created sets.
[9] *Release* command releases search result sets not needed anymore.
[10] *Livt* views Legislative Indexing Vocabulary Thesaurus online.
[11] *Expand* command combines *Livt* and *Select* commands.
[12] *Select* command creates 1 or more record sets for a specified search term.

	Pattern	*Support*	*Score*	*Density*
1	6-7-8-9-10-7	19	10.98	1.0
2	**4-14-15-6-7-8**	14	9.84	1.0
3	**22-23-22-6-7-8-9-10**	9	9.51	1.0
4	**4-5-6-7-8-9-10**	10	9.33	1.0
5	22-23-22-6-7-8	12	9.27	1.0
6	4-5-6-7-8-9	11	8.94	1.0

Table 6.8. The Qualified Maximal Patterns Discovered Using *c* (6,9,0,7) for The Interaction Pattern mining Case Study.

	Pattern	*Support*	*Score*	*Density*
1	**21-22-23-22-6-7-8**	8	8.42	1.0
2	**15-6-7-8-9-10**	8	7.75	1.0
3	7-8-9-10-7-4	8	7.75	1.0

Table 6.9. The Qualified Maximal Patterns Discovered Using *c* (6,8,0,7) That Are Not in Table 6.8 for The Interaction Pattern mining Case Study.

6.7.1.2 The Second Run

The second run was done with c = (6,8,0,7) to see what extra patterns would be discovered if less support was required. The run gave the three extra patterns shown in Table 6.9 besides those shown in Table 6.8. These extra patterns do not represent any new tasks, as they widely overlap with the three significant patterns of Table 6.8. Close examination of instances of the extra patterns revealed that the two patterns in bold enhance the current understanding of the user tasks. The first bold pattern (pattern 1) is a sub-pattern of pattern 3 in Table 6.8 but with Id 21 extra, which suggests that some instances of the corresponding task may optionally start with Id 21. The second bold pattern overlaps with pattern 2 in Table 6.8, suggesting that the corresponding task is actually the union of both patterns. These findings suggest modifying the three interaction patterns or task representations given earlier to be:

1. 4^+-5-6^+-7^+-8^+-9^+-10
2. 4^+-14-15^+-6^+-7^+-8^+-**9^+-10**
3. **[21]**-22-23-22-6^+-7^+-8^+-9^+-10

where [*n*] means that an instance of Id *n* may or may not exist. Modifications are shown in bold font.

	Pattern	*Support*	*Score*	*Density*
1	21-22-23-22-6-7-8	13	9.85	0.95
2	22-23-22-6-7-8-9-10	10	9.84	0.99
3	4-5-6-7-8-9-10	10	9.33	1.0
4	**22-23-22-6-7-8-10**	12	8.99	0.89
5	**4-5-6-7-8-10-7**	11	8.8	0.91
6	**22-23-6-7-8-9-10**	10	8.16	0.88

Table 6.10. The Qualified Maximal Patterns Discovered Using *c* (7,10,1,7) for The Interaction Pattern mining Case Study.

	Pattern	*Support*	*Score*	*Density*
1	22-23-22-6-7-8-9-10	13	10.4	0.94
2	21-22-23-22-6-7-8	17	10.34	0.9
3	7-4-14-15-6-7-8	16	9.75	0.87
4	4-5-6-7-8-9-10	12	9.61	0.96
5	**22-23-22-6-7-8-10**	13	9.18	0.88
6	6-7-8-9-10-7-4	12	9.09	0.9
7	6-7-8-9-10-21-22	13	8.92	0.86
8	7-8-7-4-14-15-6	12	8.63	0.86
9	**22-23-6-7-8-9-10**	12	8.63	0.86

Table 6.11. The Qualified Maximal Patterns Discovered Using *c* (7,12,2,7) for The Interaction Pattern mining Case Study.

6.7.1.3 The Third and Fourth Runs

The third run was done with $c = (7,10,1,7)$ to see the effect of allowing some errors in the episodes that support the pattern on the results retrieved. The minimum support was increased to limit the results set since allowing insertion errors usually increases the number of retrieved patterns significantly. The retrieved patterns are shown in Table 6.10. The three patterns in normal font were also retrieved in the previous runs. The bold patterns add significant information to the task models discovered so far. Patterns 4 and 5 have more support than patterns 2 and 3 respectively, but lack Id 9. This suggests that Id 9 is optional in these tasks. Reviewing few instances that support patterns 4 and 5 proves this, especially that Id 9 represents "Display Item (2/n or more/n) Page" as in Table 6.7. Since some items in the library catalog have only two pages of details, i.e., a first and a last item details pages (Screen Ids 8 and 10), then the related interaction patterns do not include instances of Id 9. Similarly, pattern 6 is identical to pattern 2 with Id 22 missing. Since they have the same support, one can be deceived and think that pattern 6 is a false pattern, i.e., it is the same as 2 with Id 22 considered the one spurious Id (error) permitted in this run. However, pattern 2 is supported by nine exact episodes (as shown in Table 6.8) and one approximate episode. Hence, 9 supporting episodes of pattern 6 actually have Id 22 and they do in fact support pattern 2 as well, but one episode that supports pattern 6 lacks Id 22, suggesting it is optional for the corresponding task. These findings are also emphasized by the results of the fourth and last run with $c = (7,12,2,7)$, which are shown in Table 6.11. The patterns in normal font were previously discussed in Tables 6.8 and 6.9. The patterns in bold font lead to the same conclusion as those in bold font in Table 6.10. The gray ones are spurious patterns.

6.7.1.4 The Final Results

The result of the above findings is modifying the task models discovered as shown below in bold:

1. 4^+-5-6^+-7^+-8^+-**[9^+]**-10
2. 4^+-14-15^+-6^+-7^+-8^+-**[9^+]**-10
3. [21]-22-23-**[22]**-6^+-7^+-8^+-**[9^+]**-10

The task corresponding to the first pattern of the three interaction pattern discovered {4^+, 5, 6^+, 7^+, 8^+, [9^+], 10} was discussed in subsection 6.1. It is shown in Figure 6.5, with an extra arc to reflect that Id 9 is optional. In the second task {4^+, 14, 15^+, 6^+, 7^+, 8^+, [9^+], 10}, the user starts by browsing part of the currently open library catalog. Then s/he issues a *select* command to retrieve some records from the catalog. The *select* command constructs separate subsets of results for the specified search term, each for a different search field, e.g., one for the records that have the search term in the title, one for the records that have it in the abstract, etc. Then, the user issues a *combine* command to merge some of these subsets together into one set using some logical operators. Next, s/he displays brief information about the items in this set and selects some items to display their full or partial information, using the same navigation sequence used in the first task.

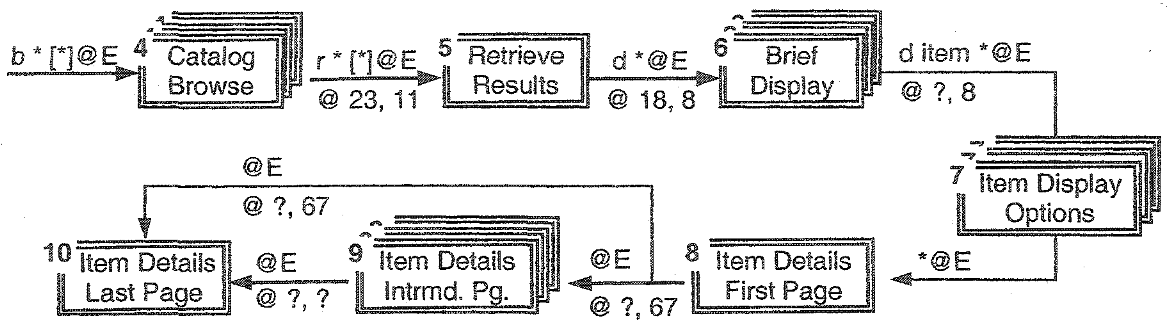


Figure 6.5. A Diagrammatic Representation of The Pattern 4^+-5-6^+-7^+-8^+-[9^+]-10, Corresponding to The Information Retrieval Task of Figure 6.1(a).

In the third task {[21], 22, 23, [22], 6^+, 7^+, 8^+, [9^+], 10}, the user starts by issuing a *livt* command. This command takes as a parameter a term that is classified by LOCIS as a subject index term, and it displays all the related, broader and narrower terms available in the Legislative Indexing Vocabulary Thesaurus of LOCIS. For example, if the user wishes to search for legislation related to drugs, but thinks it is a broad term, s/he can type *livt drugs*. The results screen will display terms like Anesthetics, Antibiotics, Antihistamines, Aspirin, Generic Drugs, Narcotics, etc. Next, the user can expand some of the displayed terms using *expand* command, creating a results set of catalog entries. Finally s/he displays the needed information as in the two other tasks.

In all three tasks the legacy system may follow alternative paths to present the results to the user, depending on how many pages of details are retrieved for the legislation of interest. In the last task, other optional steps exist as well.

6.7.2 A Comparison between IPM and IPM2

In this subsection a comparison between IPM and IPM2 in terms of their memory and time requirements is presented. In order to perform this comparison with long traces, a component, called Legacy System Trace Generator (LSTG), was added to LeNDI. The next is a description of LSTG, followed by the experiment details.

6.7.2.1 Legacy System Trace Generator (LSTG)

LSTG simulates an existing legacy system and generates traces as sequences of Ids for the purpose of testing the interaction pattern mining capabilities of LeNDI. LSTG models the navigational behavior of a legacy system user as captured in the interaction traces recorded while the user was working with the legacy system of interest. The model produced is in the form of a transition matrix whose rows and column correspond to the

Ids of the legacy system screens as given by LeNDI's clustering module. A cell, $cell_{\text{trans}}$ $[i, j]$, in this matrix contains the probability of having a transition from Id i to Id j. The probability is calculated by dividing the number of times the transition from i to j was recorded in the traces by the total number of occurrences of Id i in the traces. Practically, the transition matrix is converted to and stored as an accumulative transition matrix. A cell in this matrix is calculated as follows:

$$cell_{\text{accum}}[i, j] = \sum_{k \leq j} cell_{\text{trans}}[i,k]$$

Additionally, all possible starting screens (Ids) of the system that were recorded in the traces are stored in an array. In all the real systems we dealt with so far, there was only one start Id. Then using this model, artificial traces of arbitrary length can be generated that simulate the navigational behavior of the user whose navigation was captured in the original traces. Given the desired length, this is done as follows:

1. LSTG randomly picks, from the list of possible starting Ids, an Id, id_1,
2. LSTG generates a random number that is in the interval [0,1). Then it searches the row corresponding to id_1 in the accumulative transition matrix for the first cell that is larger than the generated number and takes the corresponding column's Id as the next Id in the trace, id_2, and
3. Then, id_3 is generated as in step 2 and so on and so forth until a trace of the required length is generated.

Note that only the transitions that occurred in the real recorded interaction traces can occur in the artificial traces. Also, the probability of such a transition in the artificial traces equals its probability in the original traces. Different transition probabilities matrices may exist for the same system, depending on the tasks being captured and modeled at the time, and hence the navigational sequences of interest that accomplish these tasks. Note that LSTG does not generate sample snapshots for the Ids generated.

6.7.2.2 Experiment Details

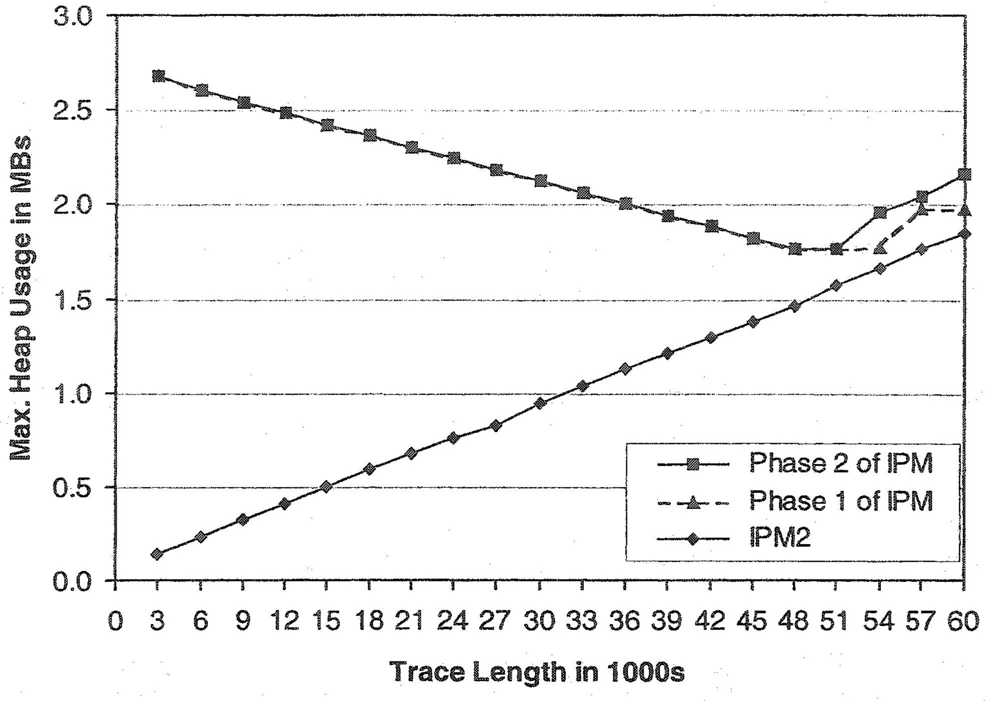
After describing how LSTG works, the specifics of the experiment follow. First, the traces of the case study of subsection 6.7.1 were fed to LSTG and the corresponding accumulative transition matrix was produced. It is shown in Table 6.12, with accumulative probabilities replaced by percentages. Due to space limitation, if a number of a consecutive cells of Table 6.12 on the same row have the same percentage, they are all merged together. For example, all cells from [20, 12] to [20,19] have the value 16.3%. Second, three criteria were selected for the experiment. The first criterion c_1 (*minLen, minSupp, maxError, minScore*) is (6, 0.5%, 0, 0), where *minSupp* is chosen to be a percentage of the trace length so that it scales up with the length of the trace generated. It is set to 0.5% since in the case study of subsection 6.7.1, the initial support used was 9, which is 0.54% of 1657, the total length of all the traces used. So, a minimum support of 0.5% would result in a comprehendible set of patterns. The second criterion, c_2, is (7, 0.5%, 1, 0). The third criterion, c_3, is (7, 0.5%, 2, 0). For each criterion, interaction pattern mining was done using IPM and IPM2 on artificial traces of length starting from 3000 till 60000, with a step 3000, that were generated randomly using LSTG. For every run, the time needed and the maximum heap used were recorded. The results of these runs are shown in Figures 6.6 to 6.8.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27
1	6.3	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	87.5	100	100	100	100	100	100	100	100	100	100	100
2	0	0	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
3	15.4	15.4	15.4	69.2	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	76.9	92.3	92.3	92.3	92.3	100	100	100
4	1.5	1.5	1.5	33.3	56.8	57.6	57.6	57.6	57.6	57.6	71.2	71.2	71.2	93.9	93.9	93.9	93.9	94.7	94.7	97.7	97.7	97.7	97.7	97.7	97.7	97.7	100
5	1.8	1.8	1.8	14.5	21.8	65.5	65.5	67.3	67.3	67.3	76.4	89.1	90.9	90.9	90.9	90.9	90.9	90.9	92.7	96.4	100	100	100	100	100	100	100
6	0	0	0	0.4	0.7	56.3	89.6	91	91	91	96.6	98.1	98.9	98.9	98.9	98.9	98.9	98.9	98.9	98.9	98.9	98.9	98.9	98.9	98.9	98.9	100
7	1	1	1	14.5	17	17	26	87	87	87.5	88.5	91	91	91	91	91	91	91	91	94	98.5	98.5	98.5	98.5	99.5	99.5	100
8	0	0	0	1.9	3.7	5	28	43.5	80.7	96.9	96.9	96.9	96.9	96.9	96.9	96.9	96.9	97.5	97.5	98.1	98.8	98.8	99.4	99.4	100	100	100
9	0	0	0	0.6	0.6	0.6	0.6	3.4	69.7	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
10	1.2	1.2	1.2	12.3	12.3	12.3	66.7	69.1	69.1	69.1	69.1	75.3	75.3	75.3	75.3	75.3	75.3	77.8	77.8	86.4	93.8	93.8	93.8	93.8	97.5	97.5	100
11	0	0	0	23.1	30.8	41.8	51.6	52.7	52.7	52.7	72.5	79.1	79.1	81.3	82.4	82.4	82.4	82.4	83.5	85.7	89.0	90.1	96.7	96.7	97.8	98.9	100
12	0	0	0	8.1	9.7	29	29	29	29	29	32.3	74.2	74.2	74.2	82.3	91.9	91.9	91.9	91.9	98.4	100	100	100	100	100	100	100
13	0	0	0	0	0	40	60	60	60	60	80	80	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
14	0	0	0	0	0	12.1	12.1	12.1	12.1	12.1	18.2	21.2	21.2	24.2	93.9	93.9	93.9	93.9	93.9	93.9	93.9	93.9	93.9	93.9	93.9	93.9	100
15	0	0	0	0	0	73.0	75.7	75.7	75.7	75.7	86.5	86.5	89.2	89.2	100	100	100	100	100	100	100	100	100	100	100	100	100
16	0	0	0	11.1	11.1	11.1	11.1	11.1	11.1	11.1	11.1	66.7	66.7	66.7	66.7	100	100	100	100	100	100	100	100	100	100	100	100
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	33.3	100	100	100	100	100	100	100	100	100	100
18	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
19	0	0	0	33.3	33.3	33.3	66.7	66.7	66.7	66.7	66.7	66.7	66.7	66.7	66.7	66.7	66.7	66.7	100	100	100	100	100	100	100	100	100
20	0	0	0	4.7	7	7	7	7	7	7	9.3	16.3	16.3	16.3	16.3	16.3	16.3	16.3	16.3	51.2	83.7	83.7	83.7	83.7	97.7	100	100
21	0	0	0	0	0	0	0	23	23	23	9.1	9.1	9.1	9.1	9.1	9.1	9.1	9.1	9.1	11.4	22.7	75	75	75	77.3	100	100
22	0	0	0	0	0	38.1	38.1	38.1	38.1	38.1	46	46	46	46	46	46	46	46	46	46	46	47.6	96.8	98.4	98.4	100	100
23	0	0	0	0	0	2.1	2.1	2.1	2.1	2.1	31.9	31.9	31.9	31.9	31.9	31.9	31.9	31.9	31.9	31.9	31.9	89.4	89.4	91.5	97.9	100	100
24	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	60	60	60	100	100
25	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	36.8	68.4	84.2	84.2	100	100
26	0	0	0	0	0	43.5	43.5	43.5	43.5	43.5	47.8	47.8	47.8	47.8	56.5	56.5	56.5	56.5	56.5	56.5	60.9	78.3	78.3	78.3	78.3	95.7	100
27	0	0	0	11.4	13.6	18.2	20.5	20.5	20.5	20.5	20.5	20.5	20.5	20.5	25	25	25	25	25	27.3	27.3	27.3	27.3	27.3	29.5	29.5	100

Table 6.12. The Accumulative Transition Matrix of LOCIS Traces of The Case Study of Subsection 6.7.1 (Probabilities are replaced by %s).

Heap usage is taken as a measure of the memory used by an algorithm while processing the given data. It is calculated using an idea similar to that explained in [Rou02] for calculating the size of a Java object. The heap usage calculated by our code is approximate and is consistent within the same experiment. But when the whole experiment was repeated three times, numbers varied significantly between the three runs, although they were still consistent within each run. Thus, the curves produced represent the relative heap requirements of IPM and IPM2 but cannot be taken as absolute measures of memory usage. The time and memory requirements of IPM and IPM2 shown in Figures 6.6 to 6.8 can be reduced if the implementation is done in C or C++ with optimization in mind. However, since LeNDI is implemented in Java, and interaction pattern mining is an offline one-time process, i.e., it needs to be performed once or a few times at most on a given data set, we focused on the correctness of the implementation rather than optimizing it.

As expected and as was intended in designing both IPM and IPM2, Figures 6.6 to 6.8 show that IPM2 needs less memory than IPM, while IPM is faster than IPM2. It is important to note that IPM and IPM2 were designed for a pragmatic reason, which is solving the interaction pattern mining problem in legacy system interaction traces. So, this experiment was done to verify the performance assumptions on which IPM and IPM2 were designed, which determine their applicability to certain problems. It is not meant to be a complete and comprehensive study of the performance of both algorithms. For such a study, different data sets with different characteristics, e.g., data size and alphabet size, need to be used plus theoretical analysis of both algorithms.



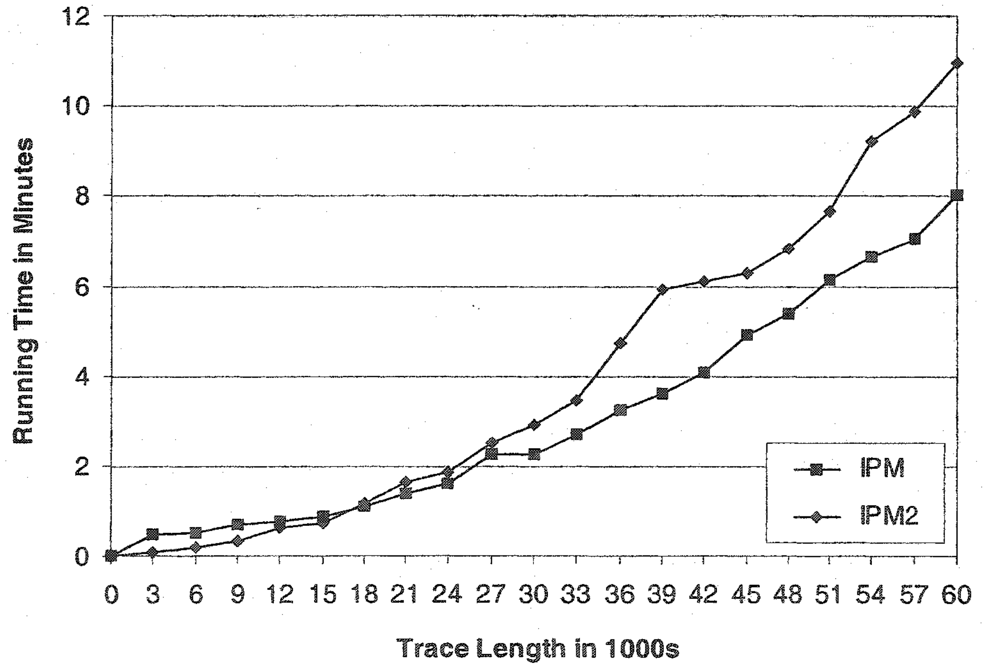
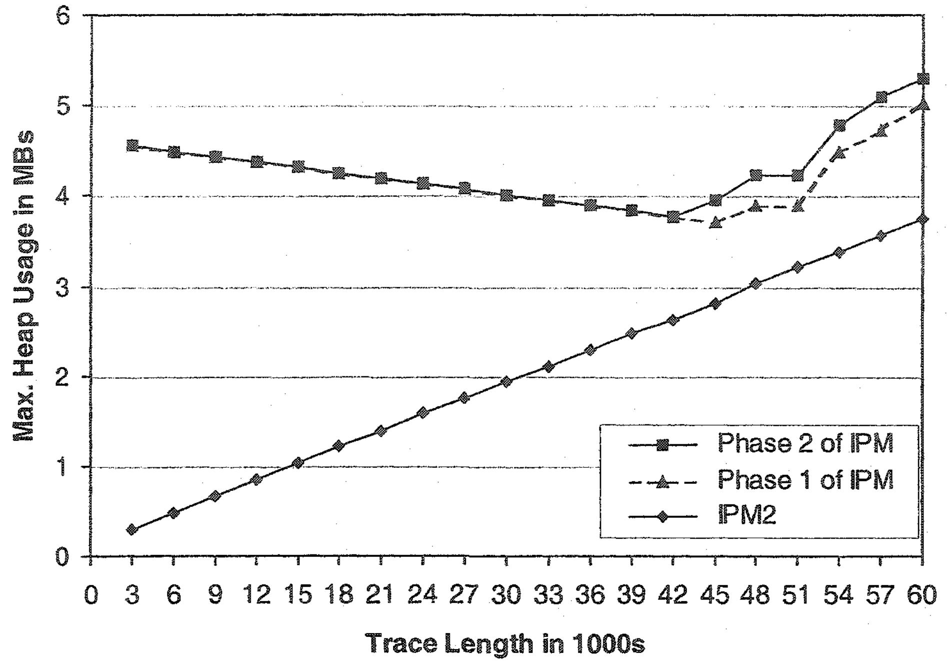


Figure 6.6. A Comparison of Memory and Time Requirements of IPM and IPM2 in The Experiment of Subsection 6.7.2 with $c_1 = (6, 0.5\%, 0, 0)$.



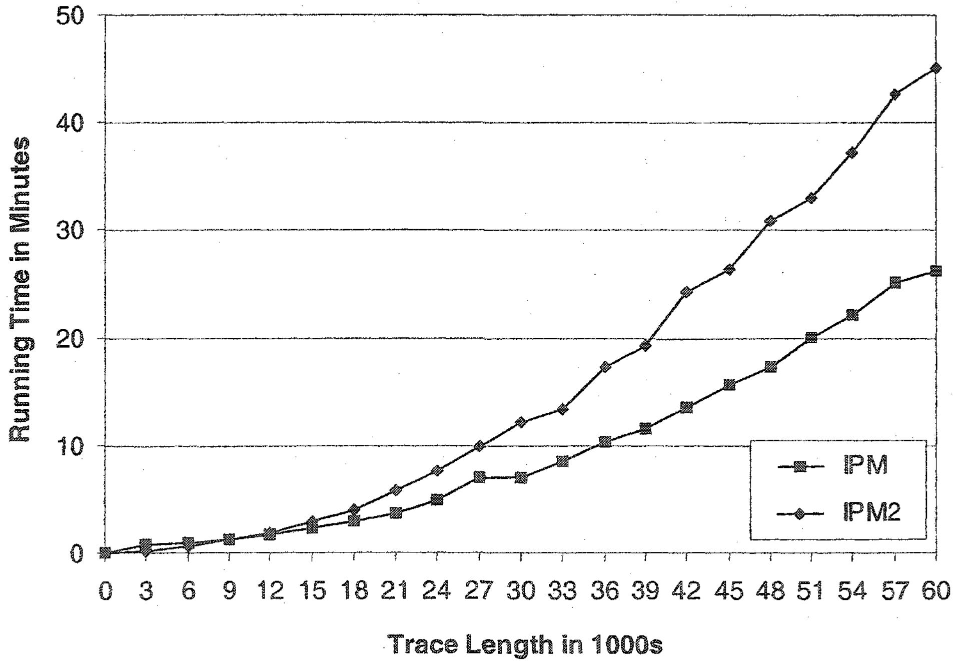
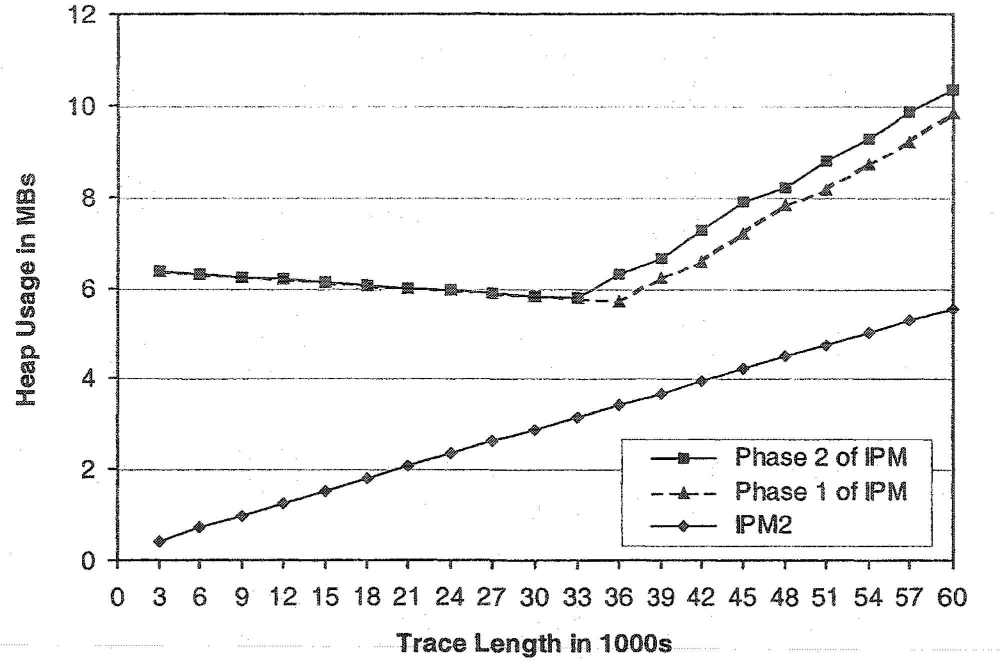


Figure 6.7. A Comparison of Memory and Time Requirements of IPM and IPM2 in The Experiment of Subsection 6.7.2 with $c_2 = (7, 0.5\%, 1, 0)$.



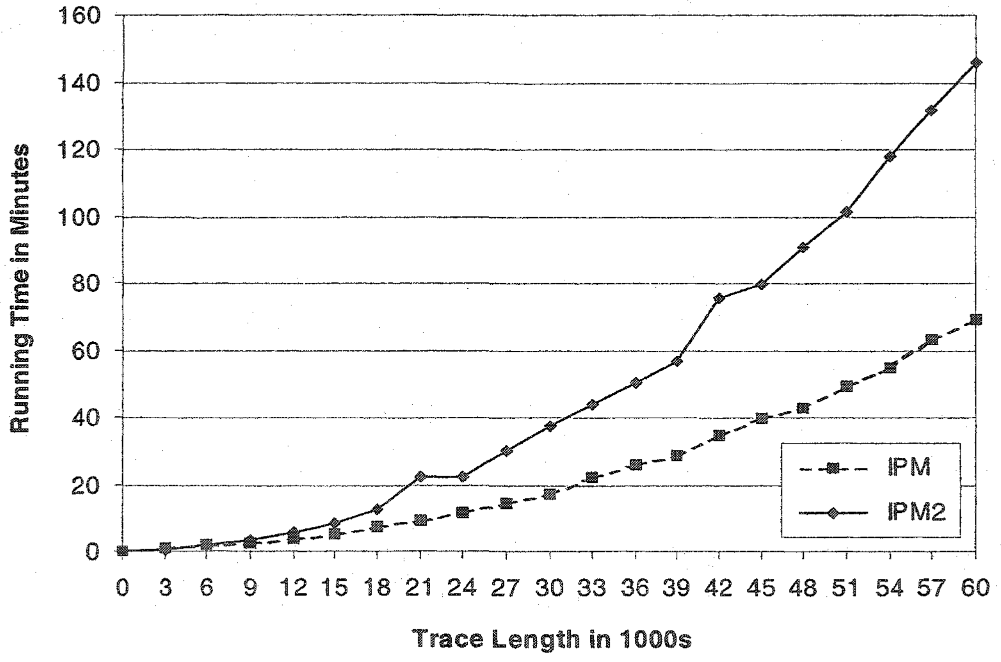


Figure 6.8. A Comparison of Memory and Time Requirements of IPM and IPM2 in The Experiment of Subsection 6.7.2 with $c_3 = (7, 0.5\%, 2, 0)$.

6.8 Discussion

The interaction pattern mining process can be considered from two viewpoints. On one hand, it is a requirement recovery process, during which, LeNDI tries to recapture the functional requirements of a legacy system as they are currently manifested and exercised not as they were originally proposed. It does this by generating hypotheses of the user tasks supported by the legacy application, which are anticipated to correspond to the system services or functionality required to be migrated. These hypotheses are verified, accepted, rejected or modified by the LeNDI analyst. In many cases, the functional requirements of a legacy system are no longer properly documented due to bug fixing, behavioral adaptations and enhancements and functionality upgrades of the system that did not back propagate to the original requirements document, assuming that such document exists in the first place. Recapturing these requirements can be useful for many reengineering, migration and program comprehension activities, other than CUI reverse engineering. Hence, the interaction patterns produced by LeNDI can be deployed in different contexts.

On the other hand this process is an instance of sequential pattern mining process: user tasks are patterns of frequently occurring episodes in the legacy CUI run-time behavior traces. In this variant of the problem, the episodes supporting the discovered patterns match only approximately. Because the users may face exceptional conditions while executing their tasks, spurious intermediate states may exist in a variety of locations in some of the episodes. Additionally, multiple paths may exist to execute one or more subtasks of a user task, resulting in slightly different navigation paths for the same task. To that end, IPM and IPM2 algorithms were developed to tackle the interaction pattern mining problem, but are general and applicable to other similar problems, e.g., discovering frequent user navigation patterns in web server logs [NSE02, ES03].

In this discussion, the focus is on interaction pattern mining from a requirements recovery viewpoint, as this is the broad application area that motivated tackling this problem. From this viewpoint, one can identify some potential uses of the interaction patterns, and/or consequently, the task models based on them. These uses include use case recovery, building user interfaces for new applications that are consistent with the existing user conceptual models, documenting interactive systems and building help or user support systems. Two examples of such potential uses are described below.

The first example is use case recovery for a legacy application. Use case models are part of the UML toolkit for object oriented system analysis and design [OMG99]. It is a widely accepted representation of the user-oriented requirements of a software system. A use case describes a sequence of interactions (activities) between a system and an external "actor" that results in the actor accomplishing a task that provides benefit to someone. An actor is a person, another software application, a piece of hardware or some other entity that interacts with the system to achieve some goal [Wie99].

For a legacy system it is not important to document the requirements that led to the original application development, but rather to capture the current uses of the application as they have evolved through continuous evolution of the application after its original deployment. These uses are the *de facto* functional requirements, as perceived by the application's current users, which are of great importance to migration activities. An interaction pattern can be looked at as a use case, which can be represented textually or by an activity diagram as shown in Figure 6.9 for the interaction pattern of Figure 6.2. Thus, the process of interaction pattern or task pattern discovery can be seen as a process of use case discovery from one type of dynamic data collected during program runs. This data is the external program behavior, represented by recorded traces of the users' dialog with the legacy user interface or simply interaction traces. However, instead of manually translating interaction patterns to use case models, it would be very beneficial to develop an automated tool support to this task.

The second example is the recovery of the task model representation of the legacy user interface in order to redesign the user interface on the same or a different platform or to build new related applications with user interfaces that are consistent with the user's conceptual model. A task model in human-computer interaction (HCI) context is a logical description of the user activities to be performed to reach a goal. In some cases alternative tasks may support achieving the same goal. A goal is either a desired modification of the state of an application or an attempt to retrieve some information from an application. This definition is not any different from what is described in this work as a task model, which is an interaction pattern enriched with semantic information.

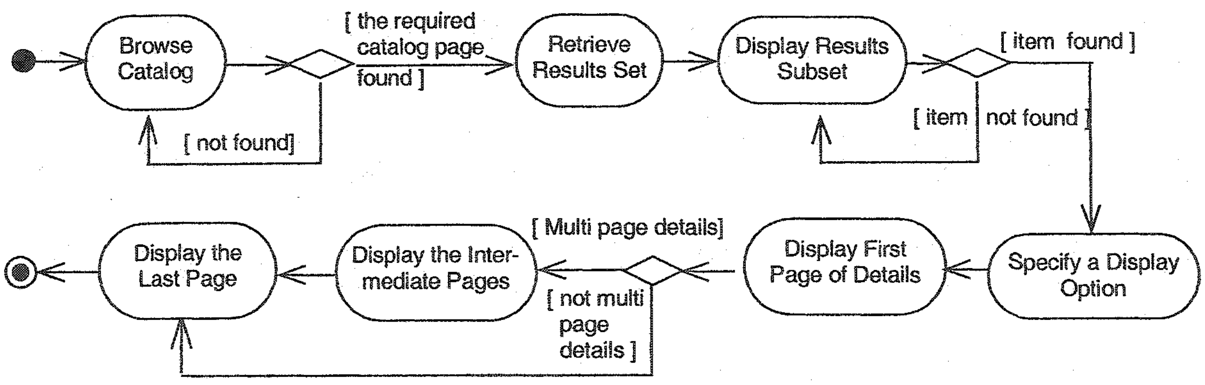
In HCI, task models are used in designing, analyzing and evaluating interactive software applications [Pat02]. Some notations were developed to describe task models in HCI context, e.g., the ConcurTaskTrees notation [LP98, Con]. Once again, it would very interesting to develop automated tool support to translate the enriched interaction patterns discovered by mining interaction traces to ConcurTaskTrees or other HCI notation.

Use Case name: Retrieving Information on a Federal Legislation
Participating actor: LOCIS User
Entry condition: The user issues a *browse* command to LOCIS
Flow of events:

1- Flip the catalog pages until the relevant page.
2- Issue a *retrieve* command to construct a results set for the chosen catalog entry.
3- Display the results set using *display* command and turn its pages until the required item is found.
4- Issue a *display item* command.
5- Specify a display option.
6- Display the item details.

Exit condition: The user retrieves the required information about the legislation he wants.

(a) A textual description of the use case.



(b) An activity diagram representation of the use case.

Figure 6.9. A Use Case Model Representing The Interaction Pattern of Figure 6.2(b).

Chapter Seven

Summary, Conclusions And Future Work

This chapter provides a summary of the work accomplished in this thesis, draws some conclusions and discusses the future directions of this work.

7.1 Summary

This thesis presents a novel method for reverse engineering the character-based user interfaces of legacy systems using recorded traces of interaction with these interfaces, as its only input. The method is implemented in a prototype tool called LeNDI. This method was developed as part of the CelLEST project for legacy system UI reengineering at the Software Engineering Research Lab., University of Alberta. The goal of the project was to develop a lightweight method for legacy user-interface reengineering, integration and Web-enabling that does not alter the legacy system's code or structure. The CelLEST UI reengineering method is two-phase, and semi-automatic. In the reverse engineering phase, a behavioral model of the legacy system character-based user interface is derived from the interaction traces. Additionally, models of the user tasks of interest are extracted from these traces. In the forward engineering phase, a reengineered UI is built. The reengineered UI interacts with the legacy system through its CUI using a host-access middleware to execute the desired task plans. The reengineered UI is task-oriented in the sense that it does not mimic the legacy user-system interaction. Instead, it encapsulates coherent user tasks or packages of functionality in suitable modern GUIs or web-interfaces, that are generated automatically using the forward engineering tool of CelLEST, Mathaino. The reverse engineering phase of CelLEST method consists of three distinct steps. The first is recording traces of interaction with the legacy system through its user interface while the users are doing their regular jobs. The second is building a behavioral state-transition model of the legacy system CUI. This model is the road map used by the new front-end to verify the identity of the legacy screen snapshots accessed to perform a user task, and hence input the appropriate inputs and extract the required outputs. The third step is mining the interaction traces for frequent segments of interaction with the legacy system, which represent hypotheses of the user tasks supported by the legacy CUI. These automatically discovered patterns are reviewed and augmented with semantic information and then used as task models to be wrapped in a new reengineered UI.

7.1.1 Trace Recording

In effect, the traces of interaction are records of the user dialog with the legacy CUI. This dialog reflects the currently active user services of the legacy system. Here, the term "active" is used to refer to the services that are still in use frequently by the system users as opposed to "inactive" or "dead" services, which are functions that are almost never used or expired due to aging. The recorded traces contain multiple usage scenarios of each service, most likely with different input data. These scenarios usually cover the active parts of the legacy CUI, specifically the screens that are frequently accessed and the user actions that are frequently entered.

The traces are recorded using a specially instrumented terminal emulator. Each trace is a sequence of screen snapshots interleaved with user actions. A snapshot consists of a presentation space (a matrix of characters received on the user's terminal) and some additional information that depends on the data transfer protocol used. For example, for the IBM 3270 data transfer protocol, LeNDI records the initial cursor location on the screen and some of the IBM 3270 field information, e.g., field location, length, and protection status (read only or read/write). A user action is a sequence of keystrokes that occurs on a snapshot as the user's response to the screen snapshot s/he sees. The current version of LeNDI can record, analyze, model and mine traces of interaction with systems that use a block-mode data transfer protocol, e.g., IBM 3270, as opposed to scroll-model protocols, e.g., IBM 5250.

By equipping enough users' desktops with LeNDI's recorder, it is relatively easy to collect a sufficient and representative number of interaction traces unobtrusively. Assuming that the recording emulators run long enough, the collected traces will cover the subsystems subject to reengineering with enough examples of screens and actions. If not, it would be easy to collect more traces that focus on covering the missing parts.

7.1.2 Behavior Modeling

Behavior modeling (chapter 5) builds a state-transition model of the legacy CUI, whose nodes represent the screens of the CUI and whose edges represent the transitions between these screens. A screen or a node reflects a behavioral state of the legacy CUI that allows a small number of user actions. Building such a model requires identifying the states and transitions of the model. Identifying the states is done in three steps: feature extraction, snapshot clustering and classifier induction. Identifying the transitions is a one step process.

Feature extraction (chapter 4) is the process of computing a feature vector for every recorded screen snapshot from its presentation space and the associated information. LeNDI employs a variety of heuristics and document analysis methods to extract a rich set of visual and other features for every snapshot. Currently, LeNDI extracts 14 single-part and multi-part features. Associated with each feature is a metric for measuring the similarity of two values of this feature. Discritization and abstraction is applied to this feature set to generate 39 single-part binary features. This binary feature set is used by LeNDI's top-down clustering algorithm that requires all binary features, i.e., whose comparison yields either 1 or 0.

After feature extraction, LeNDI clusters similar snapshots together using one of its two clustering algorithms, in order to infer what uniquely distinguishes their identity. The first algorithm is a single-path incremental clustering algorithm (subsection 5.2.1) that processes the snapshots one at a time and places each new snapshot in the most similar cluster, among the clusters available so far. If the snapshot is not similar enough to any existing cluster, then it becomes the first member of a new cluster. This algorithm is iterative and requires familiarity with the system in hand and some effort and judgement in configuring its parameters, but does not need an estimate of the number of clusters sought. The second algorithm is a top-down algorithm (subsection 5.2.2) that places all the snapshots in one cluster initially. Then it keeps decomposing the existing clusters one at time using the best-split test that minimizes the maximum internal cluster incoherence, until reaching a user-desired number of clusters or until internal cluster incoherence is below a given threshold. This algorithm is fully automated, but needs as input either an estimated number of the clusters sought or a threshold for the maximum internal cluster incoherence tolerated. This algorithm produces a decision tree that represents the best-split decision hierarchy followed to construct the output partition. The user may iteratively try different runs using different input numbers. The LeNDI analyst may choose which clustering algorithm to use depending on his familiarity with the CUI of the system under analysis. S/he can also switch from one algorithm to the other.S/he can also switch from one algorithm to the other.

After clustering, the LeNDI analyst inspects the produced partition and provides feedback regarding potential clustering mistakes by moving misclustered snapshots to their correct clusters and joining redundant clusters together. LeNDI uses this feedback to generate a classifier that is able to classify a new snapshot to one of the existing clusters using its feature vector. This classifier can then be used at runtime to recognize the identity of new snapshots as instances of the existing CUI states, and hence, to infer what actions are possible on each snapshot and to which screens they lead. Additionally, at runtime after identifying a snapshot, the new reengineered UI can apply whatever relevant input or output actions of a task plan that is being executed for the snapshot, via the host-access middleware. LeNDI has a signature-based classifier and a decision tree classifier. The later is associated with the top-down algorithm. The accuracy of the classifier induced depends mostly on the quality of the input traces, i.e., how well it covers the legacy CUI screens and behaviors.

To model the edges of the state-transition model (section 5.4), i.e., the transitions initiated from each state, LeNDI uses a model of command-language design. Currently LeNDI can model command-driven and control and function key-driven transitions but not form-filling or menu selection ones. LeNDI employs an algorithm that groups together the actions performed on the same source screen, leading to the same destination screen and analyzes them as instances of the same action. Then, it infers the different forms of this transition, if there is more than one, by analyzing these instances. LeNDI analyzes the first word in all instances first, then the second, etc. For each word, it tries to infer if it is a keyword, an option or an argument, and whether it is optional or mandatory. Additionally, LeNDI tries to infer the location of that action on the screen or the range within which it may take place.

7.1.3 Usage Pattern Mining

The purpose of this step (chapter 6) is to automate modeling of the legacy system services that would be subject to reengineering. To do so, LeNDI tries to discover patterns of frequent usage of these services in the form of frequent patterns of interaction with the legacy system that occurred in the recorded traces. Each interaction pattern is a candidate model of a system service or user task to be reengineered in terms of the interface navigation and the information exchange it implies. The patterns are enriched with additional semantic information to be ready for wrapping in a new Web-based interface or GUI. To discover these interaction patterns, data mining is applied to interaction traces through three steps. First, the traces are preprocessed to reduce their size and transform them to the format needed for the mining algorithm. Second, one of two novel interaction pattern mining algorithms of LeNDI, IPM (section 6.4) and IPM2 (section 6.5), is applied to discover the patterns. These algorithms are especially designed to suit the problem of interaction-pattern mining in recorded traces of interaction with legacy systems. IPM is a breadth first algorithm and IPM2 is a depth first algorithm. Both rely on constructing longer patterns from shorter qualified ones. Finally, the algorithm reports the patterns that meet some user criteria. This criteria define the minimum pattern length, the minimum number of occurrences, the maximum number of insertion errors allowed in the pattern instances and the pattern minimum score, according to LeNDI's scoring function. Allowing insertion errors gives the user the flexibility to accommodate user errors and unnecessary navigations like invoking help screens and/or the presence of alternative paths for some subtasks. Without allowing insertion errors, experiments showed that many useful patterns would not be recovered. Finally, the resulting patterns are reviewed by the user who may like to see sample supporting instances of each pattern to judge if it is a real pattern or just spurious repetitive navigation. Then, s/he may alter or complete the patterns chosen and then provide them as input to Mathaino to transform them to task models.

7.2 Contributions

The interaction reverse engineering method developed in this work is a novel contribution to the research in the field of reverse engineering legacy systems and to the practice of CUI reengineering and Web-enabling. The following sections describe the specific scientific and engineering contributions of this thesis.

7.2.1 Engineering a Feature Suite for Characterizing CUI screen Snapshots

The research in this thesis resulted in a suite of features for characterizing the snapshots of character-based user interface screens, and a set of corresponding document-analysis methods to extract these features from the presentation space and the hidden information of a snapshot. This suite includes three distinct feature subsets. The first is extracted from special information discovered at the periphery of the snapshot. The second is extracted from the non-visual information of the snapshot, received with IBM 3270 data streams. The third is extracted from the snapshot layout and content distribution. While the second subset is specific to IBM 3270 data streams, the first and third are general and applicable to any block-mode data transfer protocol. The effectiveness of this set in characterizing snapshots and clustering similar ones together was tested using LeNDI on real case studies with very encouraging results.

7.2.2 An Intelligent Method for Modeling the Behavior of Legacy CUIs

The second contribution is the invention of a novel semi-automated method for modeling the behavior of a legacy CUI in the form of a state-transition model, by reverse engineering the legacy system-user interaction. The steps of this method are:

1. Recording the user dialog with the legacy system in the form of interaction traces.
2. Extracting a feature vector for every recorded snapshot to use in the next step.
3. Clustering similar snapshots together to identify the distinct states of the legacy CUI.
4. Classifier induction to infer predicates for all distinct CUI states in order to recognize instances of these states at runtime.
5. Example-based learning of the syntax of the user actions causing transitions from one state to another.
6. Data mining of the interaction traces to discover frequent executions of the user tasks of interest for reengineering.

The invention of this method is a significant contribution to the research field of legacy system UI reverse engineering and to the state-of-the-art practice. It builds the necessary infrastructure and foundation for carrying out semi-automated CUI reengineering and Web-enabling. More on the strengths of this method comes in subsection 7.2.5.

7.2.3 Two Novel Sequential Pattern Mining Algorithms

In this thesis, two novel sequential pattern mining algorithms, IPM (section 6.4) and IPM2 (section 6.5), were developed to solve the interaction-pattern mining problem. However, they are general enough to mine other types of sequential data for frequent segments of navigation that may include a preset level of noise, which may occur anywhere within the segment. The algorithms were implemented in Java and they were applied in two different contexts. IPM is a breadth-first algorithm, while IPM2 is a depth-first algorithm. Consequently, the IPM2 is more space efficient than IPM, while IPM is more time efficient than IPM2. These complementary properties make them appropriate for different application scenarios. An experimental comparison (subsection 6.7.2) between IPM and IPM2 was conducted on long sequences of artificial data generated with a legacy system simulator designed for that purpose.

7.2.4 A Prototype Tool, LeNDI

The interaction reverse engineering method engineered in this work and its different components are implemented and evaluated in a prototype tool, called LeNDI (Legacy Navigation Domain Identifier). LeNDI is implemented in Java. It serves as a test-bed for the overall process and for the individual algorithms developed in this work. . It was used in reverse engineering a number of legacy CUIs with promising results.

7.2.5 The Strengths of Interaction Reverse Engineering

The legacy CUI reverse engineering method developed in this work is novel and distinct in several ways. First, this method employs an easy to collect, yet underutilized, input, which is interaction traces. Therefore, our interaction reverse engineering method is code-independent and does not require any modifications of the legacy system code or even the availability of the code, its documentation or the right to modify it. It is independent of the programming language used and implementation details. Hence, it is suitable for reengineering legacy systems when it is desired to keep the system running on its platform and only migrate its front-end or integrate it with other systems' front-ends, while changes to the code are undesirable. In cases when it is impossible to change the existing system, this becomes the only way to reengineer it. This approach has the limitation that only limited functionality extension and re-purposing can be done, and only according to what is offered by the legacy CUI. In cases when the source code and documentation are unavailable, interaction reverse engineering becomes a valuable method for comprehending the system for maintenance or other purposes, other than reengineering.

Second, interaction reverse engineering employs a mixture of document analysis, feature extraction, clustering, classifier induction, data mining and modeling methods, in the reverse engineering phase of CelLEST, to leverage and advance the current practices of legacy CUI reengineering. It supersedes the manual practices of screen scraping and mapping by introducing a coherent automated process that is less time and cost demanding and less error-prone. Consequently, the overall CelLEST process does not replicate the legacy system-user interaction with different widgets in new platforms, but adopts a task-centered approach that encapsulates interesting behavioral segments in new UI front-ends on different platforms.

Third, the method is lightweight in terms of the skills it assumes. It needs moderate analysis skills and fair understanding of the system under analysis as opposed to the solid software development and programming skills and expert understanding of the legacy system that current practices need. Although we did not conduct a formal usability experiment, we can report that after 2 or 3 hours of training a junior member of CelLEST project team, who is a summer student, could actually use LeNDI to record traces of interaction with an IBM 3270 legacy system and reverse engineer and model its CUI.

Fourth, interaction reverse engineering constructs a high-level, intermediate abstraction of the legacy system behavior in the form of state-transition and task models. These models are used in the subsequent CelLEST forward engineering process to support abstract interaction reengineering and hence, simultaneous migration to multiple platforms.

Fifth, it is possible to reverse engineer only some portion(s) of the legacy CUI if these are the only parts that need be reengineered, comprehended and/or modeled. Also, it is possible to do staged legacy CUI reengineering using CelLEST, starting by the services that are most desired to be reengineered, etc. In other words part of being a lightweight engineering method, CelLEST and consequently, its reverse engineering phase, are incremental methods as opposed to big bang reengineering methods, which are risky and expensive.

Sixth, experiments for testing and evaluating LeNDI and the underlying methods have given very promising and encouraging results. They have provided ample evidence of the usefulness and applicability of the methods proposed in this work, although there is still room for future improvements and enhancements.

Finally, because exactly this work has been motivated by a partnership with an industrial sponsor and its methodology is inspired by industrial practices in the area, we believe that our interaction reverse engineering method can potentially have an impact to the legacy migration and CUI reengineering practice.

7.3 Future Work

Subsections 4.8, 5.6 and 6.8 discussed in details the future work for improving and enhancing LeNDI and the underlying methods, and for extending the use of individual methods to other areas. This section discusses possible extensions of the entire interaction-based CUI reverse engineering process in two orthogonal directions. The first is using interaction-based legacy CUI reverse engineering for purposes other than UI reengineering. The second is extending it to different types of interaction, other than interaction with legacy systems, and hence broadening the application spectrum of this method.

7.3.1 Other Applications of CelLEST Legacy CUI Reverse Engineering Method

The research and ideas presented in this thesis can be utilized beyond their use in CelLEST project. In the future, some of these other uses will be explored, especially the ones presented below.

First, the analysis done to reverse engineer a legacy CUI is a form of dynamic analysis [SS02], which aims to model and understand the external dynamic or run-time behaviour of the legacy system. Mostly, dynamic analysis focuses on the internal behavior of the software during run-time, e.g., flow of control, memory utilization, function entry and exit data, count of executed instructions, etc. [RR01], etc. There is a growing interest in combining static and dynamic analysis of legacy and large software systems for better program understanding, visualization and other purposes [IWPC01]

[LLS01]. It would be interesting to see how external behaviour analysis, in the form of interaction reverse engineering, can be combined with static or code analysis of legacy systems or with dynamic analysis of the internal program behavior for better and more complete program understanding.

Second, the state-transition model and interaction patterns generated for a legacy system can aid the process of re-documenting an existing system or developing a help system (documented or electronic). This can be done with a pragmatic approach that focuses on documenting the currently "active" or "usable" functions of the system, from a user perspective. Interaction patterns can be translated, after some editing by an expert on the system, to "how to" subsections in the new user document.

Third, interaction pattern discovery, as explained in chapter 6, can be seen as a form of requirements recovery. It can be used to infer use cases for systems that were developed before the advent of UML. Or, it can be used to recapture the current uses of the system as its *de facto* functional requirements for the purpose of aiding system migration, building a new system or extending the system with new subsystems that respect the users' conceptual models of the tasks they perform.

Fourth, it is possible to use interaction reverse engineering as a means for "objectifying" or "APIing" a legacy system by creating a new API for it via screen mapping as briefly introduced in subsection 2.3.5. In such a case, a task model can be encapsulated in a function or a procedure that implements the corresponding task plan and executes it whenever it is invoked. Then, the outputs of this plan are not presented to the user directly via a new GUI or Web-UI, but are consumed by the calling program. In an object-orientated context, a group of related legacy system services can be encapsulated in a class, with the corresponding task models encapsulated in methods. This way, it is possible that some or all of the legacy system services are integrated with other programs in creating bigger applications, with minimal effort.

7.3.2 Reverse Reengineering Different Types of Interaction

Interaction reengineering is a broad approach for legacy system reengineering that is not necessarily related to legacy CUIs. It simply means reengineering the way the users of a system interact with the system, without necessarily reengineering the code of the system, although code reengineering may be needed depending on the goals to be achieved. In the case of a legacy CUI, as shown in this thesis, this is done be understanding and modeling the current CUI and the user tasks of interest and then forward engineering these tasks. To reengineer other kinds of interaction, one would still need similar reverse engineering and forward engineering steps, which may differ in their details from the case of legacy CUIs depending on the context.

A potentially interesting use of interaction reengineering is to extend it to legacy applications that use scroll-mode data transfer protocols, like VT 100 and IBM 5250. The challenge in this case is defining the elements of the state-transition model of the legacy CUI and then identifying them from the recorded traces. Defining the elements of the state-transition model means characterizing what constitutes a behavioral state of the system that corresponds to a legacy screen in the case of IBM 3270 and also what an edge would be in this case.

We have applied interaction reengineering for lightweight web site run-time reengineering by introducing on-the-fly URL recommendations [ES03]. The target of that work is focused web sites, which are web sites that support an ongoing process and offer information essential to that process, e.g., web sites of university courses. Users navigate such sites in a consistent task-driven (as opposed to data-driven) way that reflects the tasks of the underlying process. In the reverse engineering step, we applied interaction pattern mining using IPM2 to discover frequent user navigation patterns from server logs of the first three working days of the week. In the forward engineering step, these patterns are used to generate URL recommendations for students navigating the web site in the last two working days of the week. The choice of the length of the logs to use for pattern generation and of the period during which these patterns would be recommended is optional. Recommendation is done by instrumenting the server to use dynamic page re-writing with hidden fields for two tasks. First, it is used to keep track of the client identity using embedded session-specific Ids. Second, it is used to recommend some URLs to the user based on her/his navigation history. This is done by matching the user's recent navigation history with the prefixes of the collected patterns and offering the suffixes of the relevant patterns, or some of them based on a selection criterion, as URL recommendations for subsequent navigation. During dynamic re-writing, these URLs are embedded in the HTML page before forwarding it to the client.

While the application described in [ES03] seems to be quite different from the work of this thesis, in essence we applied the same process in both cases. A legacy system here corresponds to a web site there. A recorded trace of interaction with the legacy system corresponds to a web log. Frequent user tasks performed in interaction with a legacy CUI correspond to frequent web site navigation segments. The purpose of forward engineering phase was different. In [ES03] it was simply to save some navigation steps by predicating where the user may like to go based on her/his navigation history. In this work, no code is touched, while in [ES03] minimal run-time HTML page re-writing is required in order to identify the clients and to insert recommendations in the web pages received.

Interaction reengineering can also be applied to window-based applications or GUI driven applications (GDAs) in general. In principle, it would be possible to monitor the sequences of events occurring in the service of user tasks and then inferring some model or plan of this task. Then it would be possible to encapsulate this plan in a class method in order to replay it by invoking the method from other programs. Thus the user interaction with the GDA can be reengineered and/or the services of the GDA can be integrated with other applications or used to build bigger applications.

References

[AFMT95] G. Antoniol, R. Fiutem, E. Merlo and P. Tonella, *Application and User Interface Migration From Basic to Visual C++*. In Proc. of the Int. Conf. on Software Maintenance (ICSM), pg. 76-85, 1995.

[AGL98] R. Agrawal, D. Gunopulos and F. Leymann, *Mining Process Models from Workflow Logs*. In Proc. of the 6th Int. Conf. on Extending Database Technology (EDBT), pg. 469-483, 1998.

[AIS93] R. Agrawal, T. Imielinski and A. Swami, *Mining Association Rules between Sets of Items in Large Databases*. In Proc. of the 1993 Int. Conf. on Management of Data (SIGMOD 93), pg. 207-216, 1993.

[Ake00] L. Akers, *Web-enabling Legacy Applications – An Overview for VSE Users*. VSE/ESA Software Newsletter, IBM, Third/Fourth Quarter, 2000.

[Amb00] S. Ambler, *Legacy Integration Techniques for Java Applications: How to Reuse Your Legacy Investments within Java Applications*. IBM developerWorks Journal, IBM, Nov. 2000.

[AS94] R. Agrawal and R. Srikant, *Fast Algorithms for Mining Association Rules*. In Proc. of the 20th Int. Conf. on Very Large Data Bases (VLDB), pg. 487-499, 1994

[AS95] R. Agrawal and R. Srikant, *Mining Sequential Patterns*. In Proc. of the 11th Int. Conf. on Data Engineering (ICDE), pg. 3-14, 1995.

[Att00] Attachmate, *Repurposing Legacy Applications for the Web: Screen-Based Access in Perspective*. A White Paper, Attachmate Corporation, 2000.

[BB01] D. Berman and K. Bregar, *Don't Replace -- Extend: Why Leveraging Your Legacy Systems Is the Way to Go*. Enterprise Systems, June 2001.

[BB94] A. Bairoch and P. Bucher, PROSITE: Recent Developments. Nucleic Acids Research, vol. 22, pg. 3583-3589, 1994.

[BCB00] J. Baixeries, G. Casas and J. Balcazar, *Frequent Sets, Sequences, and Taxonomies: New, Efficient Algorithmic Proposals*. Report Number LSI-00-78-R, El departament de Llenguatges i Sistemes Informàtics, Universitat Politècnica de Catalunya, Spain, Dec. 2000.

[BDVHHP00] Brejova, B., DiMarco, C., Vinar, T., Hidalgo, S. R., Holguin, G. and Patten, C. *Finding Patterns in Biological Sequences*. Unpublished project report for CS798G, University of Waterloo, Fall 2000.

[BFM02] B., Braswell, G. Forshay and J. Martinez, *IBM Web-to-Host Integration Solutions*. Redbooks Series, IBM, Jan. 2002.

[BL97] A. Blum and P. Langely, *Selection of Relevant Features and Examples in Machine Learning*. Artificial Intelligence, vol. 97, no.1-2, pg. 245-271,1997.

[BS02] R. Biuk-Aghai and S. Simoff, *Assisting the Design of Virtual Work Processes via On-line Reverse Engineering*. In Proc. of the 35th Hawaii Int. Conf. on System Sciences, pp. 58-67, 2002.

[BSTWW99] J. Bergey, D. Smith, S. Tilley, N. Weiderman and S. Woods, *Why Reengineering Projects Fail.* Technical Report CMU/SEI-99-TR-010, Software Engineering Institute, April 1999.

[CCDD01] G. Canfora, A. Cimitile, A. De Lucia and G. Di Lucca, *Decomposing Legacy Systems into Objects: An Eclectic Approach*. Information and Software Technology, vol. 43, no. 6, pg. 401-412, 2001.

[Cel] Celcorp, *www.celcorp.com.*

[Cel99] Celcorp. *CelEngineer User's Guide – Evaluation Version 2.0.* Celcorp, 1999.

[Cha98] R. Chadha, *Integration of Web with Legacy Systems Through Java Applets and Distributed Objects*. In Workshop on Compositional Software Architectures, 1998.

[Coh94] W. Cohen, *Recovering Software Specifications with Inductive Logic Programming*. In Proc. of the 12th National Conf. on Artificial Intelligence, vol. 1, pg. 142-148, 1994.

[Con] The ConcurTaskTrees Environment Version 1.5.6. Available at http://giove.cnuce.cnr.it/ctte.html.

[Cri01] R. Crigler, *Use Screen Mapping For Wireless Access to Legacy Enterprise Data.* Enterprise Application Integration (EAI) Journal, Aug., 2001.

[CWSR00] S. Comella-Dorda, K. Wallnau, R. Seacord and J. Robert, *A Survey of Legacy System Modernization Approaches*. Technical Note: CMU/SEI-2000-TN-003, Software Engineering Institute, 2000.

[D95] R. Dannelly, *Reverse Engineering X Window System based Graphical User Interface Source Code*. Ph.D. Dissertation, Auburn University, Dec. 1995.

[DFD00] G. Di Lucca, A. Fasolino, and U. De Carlini, *Recovering Use Case Models from Object-oriented Code: a Thread-based Approach*. In Proc. of the 7th Working Conf. on Reverse Engineering (WCRE), pg.108-117, 2000.

[EISSM01] M. El-Ramly, P. Iglinski, E. Stroulia, P. Sorenson and B. Matichuk, *Modeling the System-User Dialog Using Interaction Traces*. In Proc. of the 8th Working Conf. on Reverse Engineering (WCRE), pg. 208-217, 2001.

[ES03] M. El-Ramly and E. Stroulia, *Web-usage Mining and Run-time URL Recommendation for Focused Web Sites: A Case Study*. Journal of Software Maintenance and Evolution: Research and Practices, 2003. (accepted)

[ESS02a] M. El-Ramly, E. Stroulia, and P. Sorenson, *Mining System-User Interaction Traces for Use Case Models*. In Proc. of the 10th Int. Workshop on Program Comprehension (IWPC), 2002.

[ESS02b] M. El-Ramly, E. Stroulia and P. Sorenson, *Recovering Software Requirements from System-user Interaction Traces*, In Proc. of the 14th Int. Conf. on Software Engineering and Knowledge Engineering (SEKE'02), 2002.

[ESS02c] M. El-Ramly, E. Stroulia and P. Sorenson, *Interaction-Pattern Mining: Extracting Usage Scenarios from Run-time Behavior Traces*. In Proc. of the 8th ACM SIGKDD Int. Conf. on Knowledge Discovery and Data Mining (KDD 2002), 2002.

[Flo99] A. Floratos, *Pattern Discovery in Biology: Theory and Applications*. Ph.D. Thesis, Department of Computer Science, New York University, Jan. 1999.

[FOLD96] D. Howe (Editor), *The Free On-line Dictionary of Computing*. Available at www.foldoc.org.

[GBP02] M. Grechanik, D. Batory and D. Perry, *Integrating and Reusing GUI-Driven Applications*. In Proc. of the Int. Conf. on Software Reuse (ICSR), pg.1-16, 2002.

[Gold98] N. Gold, *The Meaning of Legacy Systems*. Technical Report 7/98, Dept. of Computer Science, Durham University, UK, 1998.

[HOLLIS] Harvard OnLine Library Information System (HOLLIS). The IP address of its public IBM 3270 connection is hollis.harvard.edu.

[Hor98] E. Horowitz, *Migrating software to the World Wide Web*. IEEE Software, vol. 15, no. 3, pg. 18-21, 1998.

[IBM99] IBM, *Screen Customizer Version 2.0.60: Getting Started*. IBM, 1999.

[IWPC01] E. Stroulia and T. Systa (Chairs), *Structure-Behavior-Function Program Understanding*. A Working Session at the 9th Int. Workshop on Program Comprehension, 2001.

[JM00] M. Jugel and M. Meißner, *The Javatm Telnet Application/Applet, version 2.0*. http://javassh.org/download/2.0/index.html, 2000

[Jon96] I. Jonassen, *Methods for Finding Motifs in Sets of Related Biosequences*. Dr. Scient Thesis, Dept. of Informatics, Univ. of Bergen, 1996.

[Kap01] R. Kapoor, *Device-Retargetable User Interface Reengineering Using XML*. Technical Report TR01-11, Department of Computing Science, University of Alberta, Aug. 2001.

[KD99] T. Kieninger and A. Dengel, *The T-Recs Table Recognition and Analysis System*. Lecture Notes in Computer Science 1655, Springer, pg. 255-269, 1999.

[Kie98] T. Kieninger, *Table Structure Recognition Based on Robust Block Segmentation*. Document Recognition V, pp. 22-32, 1998.

[Kon00] L. Kong, *Legacy Interface Migration: From Generic ASCII UIs to Task-Centered GUIs*. M.Sc. Thesis, Department of Computing Science, University of Alberta, Canada, 2000.

[KS96] D. Koller and M. Sahami, *Toward Optimal Feature Selection*. In Proc. of the 13th Int. Conf. on Machine Learning (ICML), pg. 284-292, 1996.

[KS01] R. Kapoor and E. Stroulia, *Simultaneous Legacy Interface Migration to Multiple Platforms*. In Proc. 9th Int. Conf. on Human-Computer Interaction, vol. 1, pg. 51-55, 2001.

[KSM99] L. Kong, E. Stroulia, and B. Matichuk, *Legacy Interface Migration: A Task-Centered Approach*. In Proc. 8th Int. Conf. on Human-Computer Interaction, pg. 1167-1171, 1999.

[Lan00] G. Langan, *From Legacy to the Web*. Enterprise Application Integration (EAI) Journal, Jan. 2000.

[LAQ99] K. Liu, A. Alderson, and Z. Qureshi, *Requirements Recovery from Legacy Systems by Analysing and Modelling Behaviour*. In Proc. Int. Conf. on Software Maintenance (ICSM), pg. 3-12, 1999.

[LBS94] Z. Liu, M. Ballantyne and L. Seward, *An Assistant for Re-Engineering Legacy systems*. In Proc. of the 6th Innovative Applications of Artificial Intelligence Conf., pg. 95-102, 1994.

[LHHP96] J. Liang, J. Ha, R. Haralick and I. Phillips, *Document Layout Structure Extraction Using Bounding Boxes of Different Entities*. In Proc. of the 3rd IEEE Workshop on Applications of Computer Vision, pg. 278-283, 1996.

[LLS01] W. Löwe, A. Ludwig and A. Schwind, *Understanding Large Software Systems – Static and Dynamic Aspects*. In Proc. of the 17th Int. Conf. on Advanced Science and Technology, (ICAST'01), 2001.

[LOCIS] The Library of Congress Information System (LOCIS). The IP address of its public IBM 3270 connection is 140.147.254.3 or locis.loc.gov.

[LP98] A. Lecerof and F. Paternò, *Automatic Support for Usability Evaluation*. IEEE Transaction on Software Engineering, vol. 24, no. 10, pg. 863-888, 1998.

[Mir96] B. Mirkin, *Mathematical Classification and Clustering*. Kluwer Academic Publishers, 1996.

[MIRLYN] Michigan Research Library Network (MIRLYN). The IP address of its public IBM 3270 connection is mirlyn.lib.umich.edu.

[MRS94] M. Moore, S. Rugaber and P. Seaver, *Knowledge-based User Interface Migration*. In Proc. of the Int. Conf. on Software Maintenance (ICSM), pg. 72-79, 1994.

[MTV97] H. Mannila, H. Toivonen and A. Verkamo, *Discovery of Frequent Episodes in Event Sequences*. Data Mining and Knowledge Discovery, vol.1, no. 3, pg. 259-289, 1997.

[NSE02] N. Niu, E. Stroulia and M. El-Ramly, *Understanding Web Usage for Effective Dynamic Web-Site Adaptation*. In the Proc. of the 4th Int. Workshop on Web Site Evolution (WSE 2002), 2002.

[OMG99] OMG, *The OMG Unified Modeling Language Specification,* version 1.3. Object Management Group, 1999.

[PA97] C. Phanouriou and M. Abrams, *Transforming Command-Line Driven Systems to Web Applications*. Computer Networks and ISDN Systems, vol. 29, no. 8, pg. 1497-1505, 1997.

[Par94] D. Parnas, *Software Aging*. In Proc. of the 16th Int. Conf. on Software Engineering, pg. 279-287, 1994.

[Pat02] F. Paternò, *Task Models in Interactive Software Systems*. In Handbook of Software Engineering and Knowledge, vol. I, World Scientific Publishing Co., pg. 817-836, 2002.

[PRSV97] C. Plaisant, A. Rose, B. Shneiderman and A. Vanniamparampil, *Low Effort High Payoff User Interface Reengineering*. IEEE Software, vol. 14, no. 4, pg. 66-72, 1997.

[PZKM99] P. Patil, Y. Zou, K. Kontogiannis and J. Mylopoulos, *Migration of Procedural Systems to Network-Centric Environments*. In Proc. of Center of Advanced Studies Conference (CASCON'99), pg. 68-82, 1999.

[Qui93] J. Quinlan, *C4.5: Programs for Machine Learning*. Morgan Kaufmann, 1993.

[REGS00] P. Rayson, L. Emmet, R. Garside and P. Sawyer, *The REVERE Project: Experiments with the Application of Probabilistic NLP to Systems Engineering*. In Proc. of the 5th Int. Conf. on Applications of Natural Language to Information Systems, pg. 288-300, 2000.

[Rij79] C. van Rijsbergen, *Information Retrieval*. Butterworths, London, UK, 1979.

[RMB00] W. Ruh, F. Maginnis and W. Brown, *Types of Integration*. In "Enterprise Application Integration: A Wiley Tech Brief", John Wiley & Sons, Oct. 2000.

[Rou02] V. Roubtsov, *Java Tip 130: Do you know your data size?* In JavaWorld (www.javaworld.com/javaworld/javatips/jw-javatip130.html), August, 2002

[RR01] S. Reiss and M. Renieris, *Encoding Program Executions*. In Proc. of the 23rd Int. Conf. on Software Engineering (ICSE'01), pg. 221-230, 2001.

[Sch99] B. Schneiderman, *Designing the User Interface*. Addison-Wesley, 1999.

[SCT02] C..Sorzano, J. Carazo and O. Trelles. *Command Line Interfaces can Be Efficiently Brought to Graphics: COLIMATE (The COmmand LIne MATE)*. Software: Practice & Experience, vol. 32, no 9, pg. 873-887, 2002.

[SEIS03] E. Stroulia, M. El-Ramly, P. Iglinski and P. Sorenson, *User Interface Reverse Engineering in Support of Interface Migration to the Web*. Automated Software Engineering, vol.10, no. 3, 2003.

[SEKSM99] E. Stroulia, M. El-Ramly, L. Kong, P. Sorenson, and B. Matichuk, *Reverse Engineering Legacy Interfaces: An Interaction-Driven Approach*. In Proc. of the 6th Working Conf. on Reverse Engineering, pg. 292-302, 1999.

[SES02] E. Stroulia, M. El-Ramly and P. Sorenson, *From Legacy to Web through Interaction Modeling*. In Proc. of the Int. Conf. on Software Maintenance (ICSM 2002), pg. 320-329, 2002.

[SESP00] S. Stroulia, M. El-Ramly, P. Sorenson, R. Penner, *Legacy Systems Migration in CelLEST*. Short Research Demonstration, In the Proc. of the 22nd Int. Conf. on Software Engineering, pg. 790, 2000.

[SK02] E. Stroulia and R. Kapoor, *Reverse Engineering Interaction Plans for Legacy Interface Migration*. In Computer Aided User-Interface Design, 2002.

[SLGSH92] S. Srihari, S. Lam, V. Govindaraju, R. Srihari and J. Hull, *Document Understanding: Research Directions*. Technical Report CEDAR-TR-92-1, Center of Excellence for Document Analysis and Recognition State University of New York, 1992.

[Sne00] H. Sneed, *Accessing Legacy Mainframe Applications via the Internet*. In Proc. of the 2nd Int. Workshop on Web Site Evolution (WSE'2000), 2000.

[SP99] J. Shao and J. Pound, *Extracting Business Rules from Information Systems*. BT Technical Journal, vol. 17, no. 4, 1999.

[SS02] E. Stroulia and T. Systä, *Dynamic Analysis for Reverse Engineering and Program Understanding*. Applied Computing Review, vol. 10, no. 1, pg. 8-17, 2002.

[TH99] V. Tzerpos and R. Holt , *MoJo: A Distance Metric for Software Clusterings*. In Proc. of the 6th Working Conf. on Reverse Engineering, pg. 187-195, 1999.

[TLRH98] Y. Tan, D. Lindquist, T. Rowe and J. Hind, *IBM eNetwork Host On-Demand: The Beginning of a New Era for Accessing Host information in a Web Environment*. IBM System Journal, vol. 37, no. 1, pg. 133-152, 1998.

[TS99] K. Tucker and R. Stirewalt, *Model Based User-interface Reengineering*. In Proc. of the 6th Working Conf. on Reverse Engineering (WCRE), 1999.

[Vij02] V. Menon, *Visualization of Legacy Interface Behavior*. A Research Report, Department of Computing Science, University of Alberta, 2002.

[Vis01] G., Visaggio, *Ageing of a Data Intensive Legacy System: Symptoms and Remedies*. Journal of Software Maintenance and Evolution, vol. 15, no. 3, pg. 281-308, 2001.

[WAP] The WAP Forum, www.wapforum.org.

[Way99] P. Wayner, *Compression Algorithms for Real Programmers*. Morgan Kaufmann Publishers, 1999.

[Wie99] K. Wiegers, *Hearing the Voice of the Customers*. Chapter 8 in Software Requirements, Microsoft Press, 1999.

[WJD01] E. Wohlstadter, S. Jackson and P. Devanbu, *Generating Wrappers for Command Line Programs: The Cal-Aggie Wrap-O-Matic Project*. In Proc. of the Int. Conf. on Software Engineering, pg. 243-252, 2001.

[Yam00] T. Yample, *Web-based Technologies for User Interface Rejuvenation*. In Web-to-Host Connectivity, A. Guruge and L. Lindgren (Ed.), CRC Press, pg. 185-197, 2000.

[ZK99] Y. Zou, K. Kontogiannis, *Enabling Technologies for Web-Based Legacy System Integration*. In Proc. of the 1st Int. Workshop on Web Site Evolution (WSE'99), 1999.