

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600

UMI[®]

**UPADE: A Tool for Automating HCI Pattern-Oriented
Designs**

Ali Ansari

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements

for the Degree of Master of Computer Science at

Concordia University

Montreal, Quebec, Canada

April 2003

© Ali Ansari, 2003



**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

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

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

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

0-612-77985-8

Canada

Abstract

UPADE: A Tool for Automating HCI Pattern-Oriented Designs

Ali Ansari

Human Computer Interaction (HCI) patterns are a contemporary topic in the literature of user interface engineering. An HCI pattern offers proven solutions to common user problems. An HCI pattern abstracts a solution structure that is mostly described in terms of a set of collaborating design structures. Composing these design structures to develop a user interface is a tedious task. A tool that supports how to combine HCI patterns to develop user interface designs will ultimately make the analysis and design phases more efficient and more productive. In this thesis, we present a tool, named UPADE to formalize the visual presentation of HCI patterns and their relationships for the purpose of developing pattern-oriented designs. The tool develops a structural composition approach to browse, search, and edit a repository of HCI patterns stored into the database. In addition, the tool supports three different levels of abstraction, namely the Pattern view, the Design view, and the Code View. The proposed tool aims to abate disparity between the HCI pattern descriptions and their implementations. It gives a systematic methodology to glue HCI patterns and supports the integration of patterns into the user interface development life cycle.

Keywords: HCI patterns, Patterns Engineering Tools, Design Patterns, and UPADE

Table of Contents

LIST OF FIGURES.....	VI
CHAPTER 1 INTRODUCTION.....	1
1.1 PATTERN IN SOFTWARE ENGINEERING	2
1.2 PATTERNS IN HUMAN COMPUTER INTERACTION	3
1.3 HCI GUIDELINES VERSUS PATTERNS.....	6
1.4 HCI PATTERNS LANGUAGES	6
1.5 PATTERNS AS A DESIGN TOOL.....	7
1.6 THESIS CONTRIBUTIONS AND RESEARCH METHODOLOGY	8
1.7 THESIS OUTLINE.....	10
CHAPTER 2 TOOLS SUPPORT FOR PATTERN ENGINEERING	11
2.1 PATTERN ENGINEERING TOOLS IN SOFTWARE ENGINEERING	11
2.1.1 Automatic code generation from design patterns.....	12
2.1.2 The Smalltalk Refactoring Browser	13
2.1.3 PSiGene CASE tool.....	14
2.1.4 The Pattern-Lint.....	15
2.1.5 Hooks and Templates.....	16
2.2 UML-ORIENTED TOOLS	16
2.2.1 MagicDraw™.....	17
2.2.2 ModelMaker Tools.....	17
2.2.3 Framework Adaptive Composition Environment (FACE)	19
2.2.4 Pattern-Oriented Analysis and Design (POAD).....	20
2.2.5 OTW (Object Technology Workbench)	21
2.2.6 Objecteering C++ Developer.....	22
2.3 PERSPECTIVE: TOWARDS AN XML-BASED TOOL.....	24
2.3.1 PCML (Pattern and Component Markup Language).....	24
2.4 THESIS CONTEXT.....	25
2.5 SUMMARY	27
CHAPTER 3 UPADE - USABILITY PATTERNS-ASSISTED DESIGN ENVIRONMENT	28
3.1 UPADE OVERVIEW	28
3.2 UPADE FUNCTIONALITY	32
3.2.1 Browsing and searching patterns.....	34
3.2.2 Facilitating the process of capturing patterns (Edit).....	40
3.2.2.1 Appending a New Pattern to the database.....	42
3.2.2.2 UPADE Feedback.....	44
3.2.3 Support developers in creating complex designs.....	44
3.2.3.1 UPADE Pattern Composition	46
3.2.3.2 Modifying a Pattern Composition.....	49
3.2.4 Generating the code from a high level XML descriptions.....	50
3.3 UPADE SCENARIO.....	51
3.4 VALIDATION OF UPADE.....	55
CHAPTER 4 UPADE IMPLEMENTATION.....	57
4.1 UPADE FRAMEWORK ARCHITECTURE	57
4.2 JDBC	61
4.3 EXTENDED MARKUP LANGUAGE (XML)	64
4.3.1 XML as a Neutral Language for Documenting Patterns and Describing the Related Design Knowledge	64

4.3.2 XML as a Device-Independent Language for Implementing Patterns	66
4.4 SUMMARY	67
CHAPTER 5 CONCLUSION AND FUTURE WORK.....	68
REFERENCES	71

List of Figures

FIGURE 2.1: AUTOMATIC CODE GENERATION FROM DESIGN PATTERNS MAIN MENU.....	13
FIGURE 2.2: THE SMALLTALK REFACTORING BROWSER.....	15
FIGURE 2.3: THE PATTERN-LINT.....	16
FIGURE 2.4: MAGICDRAW.....	18
FIGURE 2.5: DESIGN PATTERNS REFERENCE MODELMAKER.....	19
FIGURE 2.6: PATTERN-ORIENTED ANALYSIS AND DESIGN.....	21
FIGURE 2.7: OTW.....	22
FIGURE 2.8: OBJECTEERING C++ DEVELOPER.....	23
FIGURE 2.9: UML MODELER IN OBJECTEERING C++ DEVELOPER.....	23
FIGURE 2.10: PCML (PATTERN AND COMPONENT MARKUP LANGUAGE).....	25
FIGURE 3.1: A SCHEMATIC OF THE UPADE FRAMEWORK ARCHITECTURE AND PROCESS.....	32
FIGURE 3.2: PATTERN ELEMENT ASSOCIATIONS.....	33
FIGURE 3.3: BROWSE PATTERNS BY CATEGORY NAME.....	35
FIGURE 3.4: BROWSE MENU BAR.....	36
FIGURE 3.5: PATTERN DESCRIPTION.....	37
FIGURE 3.6: SEARCH RESULT TREE.....	38
FIGURE 3.7: SEARCH RESULT PANE.....	39
FIGURE 3.8: SEARCH CRITERIA.....	39
FIGURE 3.9: EDIT MAIN PANE.....	40
FIGURE 3.10: CONTROL TOOLBAR.....	41
FIGURE 3.11: ENTER NEW PATTERN.....	43
FIGURE 3.12: ENTER NEW PATTERN.....	43
FIGURE 3.13: EDIT PATH.....	44
FIGURE 3.14: DESIGN PANE MAIN PAGE.....	46
FIGURE 3.15: COMPOSING THE PATTERNS.....	48
FIGURE 3.16: SAVE MAP DIALOG BOX.....	48
FIGURE 3.17: OPEN MAP DIALOG BOX.....	49
FIGURE 3.18: GENERATE TRANSLATOR.....	51
FIGURE 3.19: THE PATTERN LEVEL.....	52
FIGURE 3.20: THE DESIGN LEVEL.....	52
FIGURE 3.21: THE CODE LEVEL.....	53
FIGURE 4.1: A SCHEMATIC OF THE CURRENT UPADE FRAMEWORK ARCHITECTURE.....	58
FIGURE 4.2: UPADE USE-CASE DIAGRAM.....	59
FIGURE 4.3: UPADE DATABASE SYSTEM.....	60
FIGURE 4.4: JDBC DIAGRAM.....	62
FIGURE 4.5: UPADE DATABASE STRUCTURE.....	63
FIGURE 4.6: A SIMPLIFIED DTD USED BY UPADE EDITOR.....	65

Chapter 1

Introduction

In 1977, Christopher Alexander shook the architectural world with his revolutionary book “The Timeless Way of Building”. He advances that one could achieve excellence in architecture by learning and using a carefully defined set of design rules, or patterns; and though the quality of a well-designed building is sublime and hard to put into words, the patterns themselves that make up that building are remarkably simple and easy to understand [Jenifer Tidwell 1998].

Each Pattern has three-part rules, which express a relation between a certain context, a problem, and a solution. As an element in the world, each pattern is a relationship between a certain context, a certain system of forces, which occurs repeatedly in that context, and a certain spatial configuration, which allows these forces to resolve themselves. As an element of language, a pattern is an instruction, which shows how this spatial configuration can be used, over and over again, to resolve the given system of forces, wherever the context makes it relevant [Alexander C., Ishikawa S., Silverstein M. 1977]. There are many dissimilarity of Alexander's original definition of pattern. However, the main elements of describing a pattern are as follow:

- **Name:** A name for the pattern. Example: Window Place.

- **Context:** A context for the design problem. Example: Design of a residential room.
- **Forces:** a Force that requires resolution. Example: People want to sit and also be in daylight.
- **Problem:** A problem growing from the forces. Example: If all seating is away from the windows, then these forces are not resolved, and people will always be dissatisfied in one way or the other.
- **Solution:** A known solution, proven in practice. Example: Build seating into the window like the traditional window seat [Casaday G. 1997].

1.1 Pattern in software engineering

Pattern in software engineering describe general reusable solutions for recurring problems in different aspects of software development. Patterns typically stem from real world experiences and their purpose is to condense the knowledge and expertise of experienced developers. Pattern Language [Alexander C., Ishikawa S., Silverstein M. 1977] is a way of representing and accumulating knowledge of good practices in software engineering. When related patterns are woven together they form a “language” that provides a process for the orderly resolution of software development problems. Pattern languages are not formal languages, but rather a collection of interrelated patterns, though they do provide a vocabulary for talking about a particular problem. In other word, a pattern language is a collection of patterns that can solve problems in a particular domain. It may include a method for connecting patterns into whole "architectures" for the domain.

Much of the existing writing on patterns is organized as design patterns. Design patterns make it easier to reuse successful designs and architectures. Expressing proven

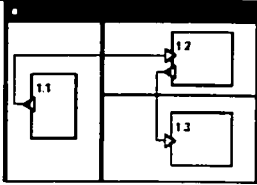
techniques as design patterns makes them more accessible to developers of new systems. Design patterns aim developer to choose design alternatives that make a system more reusable and avoid alternatives that compromise reusability. Design patterns can even improve the documentation and maintenance of existing systems by furnishing an explicit specification of class and object interactions and their underlying intent. Put simply, design patterns help a designer get a design "right" faster [Gamma E., Helm R., Johnson R. and Vlissides J. 1994].

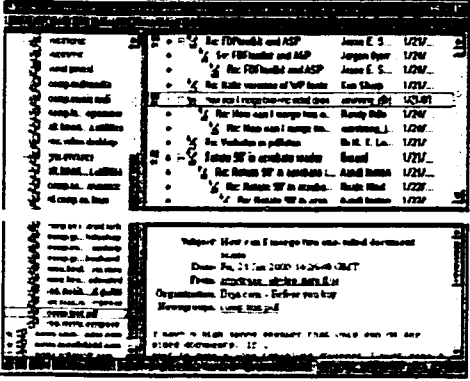
1.2 Patterns in Human Computer Interaction

The concept of Human Computer Interaction (HCI) patterns has been proposed as one step towards building more usable user interfaces. The idea is to capture information about frequently encountered user problems with software and how they can be solved. By recording the indications and remedies of typical problems, patterns can improve design reuse and usability. However, a pattern based design software can only be as good as the patterns that it is based on [Mahemoff, M.J and Johnston, L.J 1998].

For example, the following table describes the "Container Navigation" pattern proposed by Hallvard Traetteberg and Martijn van Welie [Traetteberg H. and Welie M. 2000].

Name	Container Navigation
Context	Many applications contain aggregated data, which the user must browse through. Quite often, the user wants to invoke a function taking one of the parts as input parameter.

Problem	The user needs to find an item in a collection of containers.
Forces	<ul style="list-style-type: none"> • The importance for the task that the user sees all containers and an item at the same time. • The number of items is large but only a portion of the items needs to be visible at a particular moment. • The user may need to switch from one container to the other.
Solution	 <p>Split up the screen in three areas showing the containers and the final selection. Split a window into three panes, one for the viewing a collection of containers, one for viewing a container, and one for viewing individual items. The selection of a container should determine the content of the container pane, and the selected item should determine the content of the item pane. The selections may be used as parameters to invoked functions. Each pane should be specialized to the type of content it presents. E.g. if the containers form a hierarchy a tree pane providing selection of leaf nodes could be used. The panes should be individually scrollable and resizable.</p>
Rationale	By configuring the panes according to the western way of reading (left to right, top to bottom), we support the causal relationship based on selection. By providing selection, other functionality besides navigation is supported. The layout should aid in

	<p>understanding the causality among the panes. Each pane can be tailored to the domain and user. Individually resizable panes give user freedom.</p>
<p>Examples</p>	 <p>This is Netscape's Mail/News viewer. In the left pane the set of containers (in this case newsgroup) is displayed. The other panes show the list of messages in the selected group and the selected message.</p>
<p>Known Uses</p>	<p>Microsoft Outlook; Netscape Mail/News, Eudora Mail</p>
<p>Related Patterns</p>	<p>GRID LAYOUT</p>

As we described, an HCI pattern abstracts a solution structure that is mostly described in terms of a set of collaborating design structures. Composing these design structures to develop a user interface is a tedious task. A tool that supports how to combine HCI patterns to develop a user interface design will ultimately make the analysis and design phases more efficient and more productive. However, current user interface tools do not clearly support HCI patterns as design tools.

1.3 HCI Guidelines versus Patterns

Guidelines are specific courses of action, based broadly on a set of principles. Guidelines can be construed as good practices within a general design domain, such as Windows GUI or Java Swing. They are generally more specific than principles and require less design knowledge and experience to understand and apply. For example, a guideline for implementing the user interface architecture can be like: "Provide contextual help for each choice or object that the cursor can be positioned on" [IBM 2001]. HCI design rules can be derived from the guidelines but that conversion should be performed as an integral part of the design process, serving to focus attention on critical design issues and to establish specific design requirements. An immediate response to the idea of patterns in HCI design may be that it simply replicates what has already been done through these usable guidelines. The main difference between the HCI guidelines and patterns is the unambiguous acknowledgement that patterns are more concrete. This means clearly setting the context in a hierarchical structure of patterns, showing the problem that pattern will solve, and considering a known solution to the problem. This makes patterns in general, more useful for designers than guidelines [The Brighton Usability Pattern Collection (2003)].

1.4 HCI Patterns Languages

An HCI pattern language is a collection of patterns that can solve a set of related problems in a particular domain. It may include a method for connecting patterns into

whole "architectures" for the domain. It was originally applied in town planning and architecture, but has recently been taken up by user interface and software designers.

The first HCI pattern language was developed by *Jenifer Tidwell*. The *Tidwell's* pattern language aims to support high-quality interaction between a person and a user interface including GUI, Web and mobile applications. The user may first be presented with a small range of available actions, one of which is taken; then a new set of possible actions is shown, and the user takes one of those; and so on. Therefore, *Tidwell* collect some primary patterns for actions such as: **Form, Control Panel, WYSIWYG Editor, Composed Command, And Social Space**. Using these patterns can be fairly simple: read through the language, and pick out the patterns that you see. The problem statements and forces in the pattern descriptions may help developer understand how the artifact does what it does, and what tradeoffs its designer may have been considering. In other hand, developer can use this language as a tool to help design a user interface application [Tidwell J. 1998].

1.5 Patterns as a Design Tool

An important factor in improving design quality for interactive systems is our effectiveness in capturing and communicating the essential elements of good designs. There have been many successful approaches to this task like: "study of exemplars, practice under the instruction of a master, design principles to capture the master's implicit knowledge, design rationale for organizing application of principles to cases, design guidelines making principles specific, and software toolkits embodying

guidelines” [Casaday G. 1997]. However, a new approach that has recently become outstanding in HCI development is the use of patterns to document the essentials of design solutions in a format that can be understood, learned, and applied to situations that are similar to existing cases but always subtly different.

Patterns have a larger scale of the building blocks from which to compose systems. Given a broad collection of patterns on different scales (e.g. architectural patterns vs. design patterns vs. language-idioms) it should be possible to combine and glue together patterns into a design that can be mapped to different programming languages. Pattern solutions are defined and represented by a general structure of the classes in the pattern, and an assignment of responsibilities to the participating classes. These pattern solutions can be customized for varying needs and situations. The use of design and implementation patterns is either consciously built into a system design, or pattern-like solutions get created as a result of problem solving and implementation by first principles.

1.6 Thesis Contributions and Research Methodology

In the previous sections, we briefly described how HCI patterns will help to maintenance, reuse, reverse engineering, and reimplementing of user interface applications and we also point how a usable pattern language can help software developer to design more usable and effective user interface application. The fundamental of this thesis can be stated as follows:

“How to automate the creation and composition of HCI patterns in order to facilitate the development of user interfaces while improving their usability”.

The following are the main contributions of this thesis:

- *Develop patterns-oriented design by combining HCI patterns.* To improve User interface application design, we need to stop developing applications from scratch. We need to use of existing patterns. HCI patterns make it possible to think about user interface design at high level of abstraction without any consideration to how pattern can be implemented.
- *Generating user interface code from HCI patterns.* HCI patterns address the general problem of how to design a complex interactive software artifact. However, HCI patterns only address a particular user interface design problem and it is not the code by itself. In other hand, there is a gap between the pattern description and a particular implementation, although the pattern includes code fragments in the sample code section. In contrary, some user interface developer has no difficulty to translate a pattern to the code. However, they find a complex and hard task, especially when they have to do it over and over again. This thesis, we will be describe how a HCI pattern tools can reduce this gap and make design phase more efficient and productive.

For achieving our goals, we have created a first prototype of such an environment aimed at developing user interface programs with patterns. The tool assists developers using patterns in three different levels: the Pattern Level, the design level and the code level. From just a few pieces of pattern information, the tool will create class declarations and definitions. The user interface developer then could add these codes to the rest of the application.

1.7 Thesis Outline

First, this thesis will describe existing pattern engineering tools. Then, we will describe our tool and will explain the architecture and implementation issues. This thesis is structured as follows:

Chapter 1 (this chapter) introduced the pattern history, HCI patterns, guidelines, and Pattern language. Later we described, an introduction to pattern as a design tools and finally we had thesis contributions and research methodology.

Chapter 2 overview the existing tools for supporting patterns engineering.

Chapter 3 describes our methodology for the development of pattern-oriented designs while describing UPADE. We also explain how to use the tool by using an example.

Chapter 4 demonstrates in detail the implementation of UPADE. We mainly present the UPADE architecture, overview of JDBC database, and the benefits of using XML as a tool for documenting and implementing patterns.

Chapter 5 summarizes our main conclusions and outlines the future work in the area of HCI patterns-oriented design automation.

Chapter 2

Tools Support for Pattern Engineering

In this chapter we explore the background to this research, with the aim of putting our work in context. We survey some of existing pattern tools and we will select some of conclusive points of these tools to use in our tools and at the end I will show an overview of XML based tools.

In section 2.1, we give a brief description of existing pattern engineering tools. In section 2.2, we describe UML-Oriented base tools that are using patterns. In section 2.3, we describe a perspective of an XML based tool. In section 2.4, we will explain the thesis context. And finally in section 2.5, we will give a summery about this chapter.

2.1 Pattern Engineering Tools in Software Engineering

In recent years, much activity has been concentrated on discovering and documenting HCI patterns. However, little importance has been placed on deploying these reusable designs to build a user interface application. Deploying HCI patterns to develop complex user interface is a deadly task that involves integration issues and iterative development. Yacoub and Ammar [Yacoub, S. and H. Ammar,1999] discussed

the potential benefits of using patterns as design components in the aspect of software reuse. In software reuse, code reusability provides a low percentage of effort savings but it is the most popular and common approach. Automation of this process will eventually facilitate the analysis and design phase of the user interface application [Keerthi K Arutla 2000]. In addition, there are further issues tied to pattern creation, pattern composition, and pattern usage. A tool can facilitate these kinds of activities. However, up to date, there is little existing tool that supports HCI patterns. A HCI Pattern may be used in different ways depending on user skills and their roles as a designer, developer or educator. Therefore, a tool that provides a systematic design methodology to glue HCI pattern components together is needed. For example a good HCI tool could assist:

- A pattern writer to provide HCI patterns more consistent in terminology, and more organized.
- A user interface developers to select, organize, and edit patterns relevant to a specific task, and to find relevant software components, configured appropriately using knowledge embodied in the patterns.
- A student to learn more about the problems, contexts, and solutions that occur in the field of HCI.

2.1.1 Automatic code generation from design patterns

For code generation and pattern instantiation purposes, automatic code generation from design patterns was developed to add a dimension of utility to design patterns. Users can see how domain concepts map into code that implements the pattern, and they can see how different trade-offs change the code. Once generated, the user can put the

code to work immediately. The tool displays the sections of a pattern (Intent, Motivation, etc.) in separate pages. The Code Generation page lets the user enter information with which to generate a custom implementation of the pattern. This tool is limited to system design and implementation; it does not support requirements specification, documentation, or debugging (Figure 2.1) [Budinsky, F., M. Finnie, J. Vlissides, and P. Yu 1996].

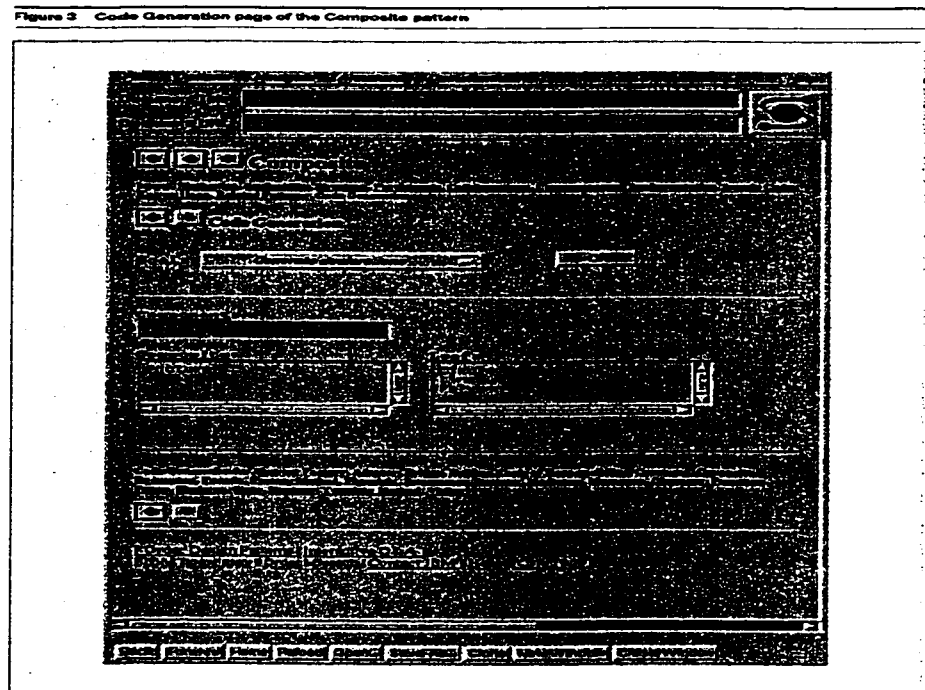


FIGURE 2.1: AUTOMATIC CODE GENERATION FROM DESIGN PATTERNS MAIN MENU.

2.1.2 The Smalltalk Refactoring Browser

“The Smalltalk Refactoring Browser” [Meijers M, 1996] assists developers using patterns in three ways:

- Generating program elements (e.g. classes, hierarchies) for a new instance of a pattern, taken from an extensible collection of "template" patterns.

- Integrating pattern occurrences by binding program elements (e.g. indicating that an existing class plays a particular role in a pattern instance)
- Checking whether occurrences of patterns still meet the invariants governing the patterns and make debugging software more efficient.

The tool is implemented in Smalltalk and has been applied to identify pattern occurrences in several non-trivial (Smalltalk) applications and to reorganize these subsequently. The fragment model/database store the program being developed as a graph of design elements such as classes, method definitions, inheritance relations, etc. The suite of design elements can be extended easily through the introduction of new fragment types. Design patterns are also design elements and represented as fragments (Figure 2.2).

2.1.3 PSiGene CASE tool

The application PSiGene - A pattern Based Component Generator for Building Simulation- CASE tool is another tool for binding patterns from a predefined catalog with class models to form the final application [Schütze M., Riegel J. P., Zimmermann G. 1999]. This method is application specific (building simulators) and it also generates the classes and methods for specific patterns from a catalog but doesn't link them together in a higher design level.

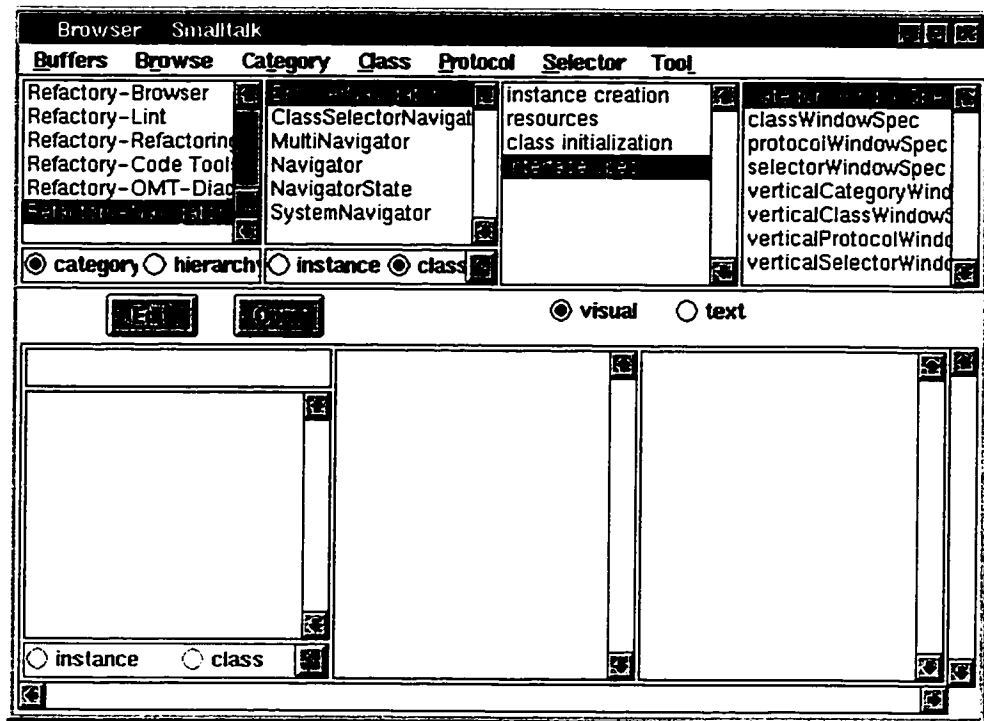


FIGURE 2.2: THE SMALLTALK REFACTORING BROWSER

2.1.4 The Pattern-Lint

The Pattern-Lint tool was designed to check the compliance of a pattern implementation [Sefika M. Sane A. Campbell R. 1996]. A set of rules is defined for each design patterns and the implementation is checked against these rules. The tool is used for analysis of systems developed from patterns but does not implement a methodology to develop applications using patterns (Figure 2.3). Currently this tool is deficient in the following features:

- This tool does not allow the user to define a pattern and save it for further reuse.
- The tool does not have a feature to save design diagrams; therefore, it could not open them in the further.

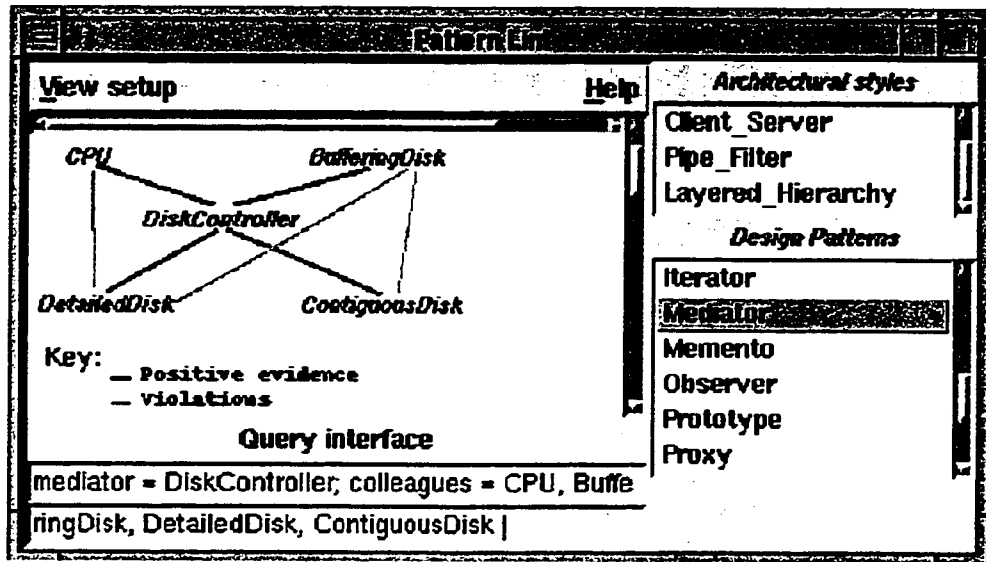


FIGURE 2.3: THE PATTERN-LINT

2.1.5 Hooks and Templates

“HOOK&TEMPLATE” is designed to recovery and program understanding by recognizing instances of design patterns semi-automatically [Pagel, B., and M. Winter, 1996]. The approach taken is specifically designed to overcome the existing scalability problems caused by many design and implementation variants of design pattern instances. “HOOK&TEMPLATE” is presented to distinguish between pattern components that will be implemented by the user and those components that are already defined for the pattern class collaboration.

2.2 UML-Oriented Tools

The choice of a good UML tool depends on what we want to do. Some of tools are very good for drawing schemas, others for generating reports, some for checking models, others for generating the code, some for reversing and few are supporting

patterns. Be aware that very few tools really support the UML specification by using the pattern. In this section, some of these tools will be explain.

2.2.1 MagicDraw™

MagicDraw UML is a modeling tool for object-oriented software development [MagicDraw, 2003]. It designed to draw all kind of diagrams and does code generation and reverse engineering for Java/C++/CORBA IDL. MagicDraw stores all model information in XML files and supports teamwork. HTML reporting facility is included for classes, use case and other diagrams are based upon XSL. Briefly, MagicDraw is a UML editor, a code-engineering tool, and An OO model analysis tool. MagicDraw can generate parts of the model automatically. You can select any object and generate the classes necessary to make it conform to a Design Pattern (Figure 2.4).

2.2.2 ModelMaker Tools

ModelMaker represents a way to develop classes and component packages for Borland Delphi (1-6) [ModelMaker ,2002]. ModelMaker is a two-way class tree oriented productivity, re-factoring and UML-style CASE tool specifically designed for generating native Delphi code. It has been used to create classes for both real-time technical and database type applications. ModelMaker is a reverse engineering tool. ModelMaker supports drawing a set of UML diagrams and from that perspective it looks much like a traditional CASE tool. Renaming a class or changing its ancestor will immediately propagate to the automatically generated source code. ModelMaker also support design patterns. A number of patterns are implemented as 'ready to use' active agents (Figure 2.5).

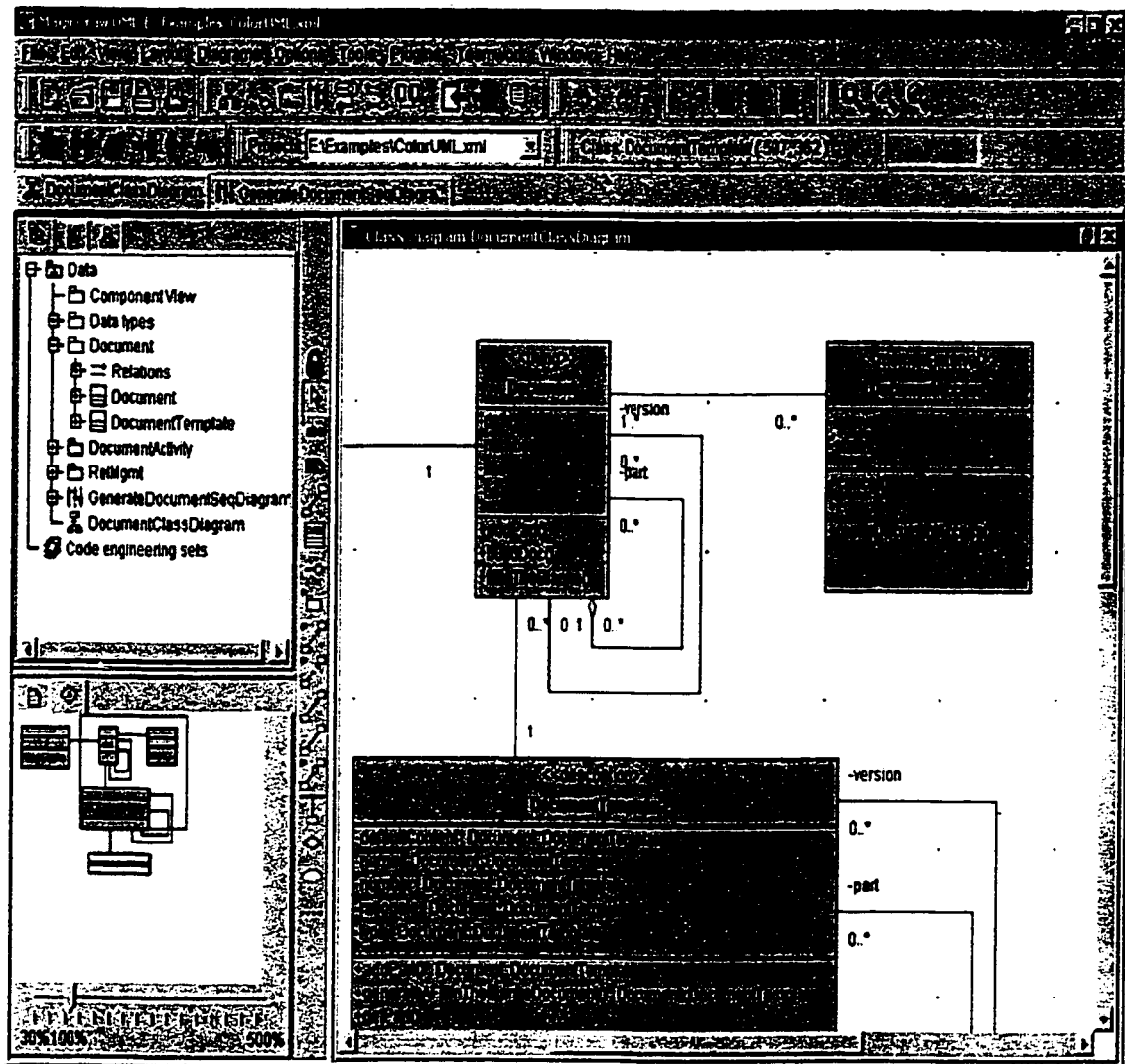


FIGURE 2.4: MAGICDRAW

A drawback to ModelMaker is that software developer are usually locked into a modal dialog box -typing and remembering class and method names is tedious and error-prone, unless a namespace browsing facility is provided.

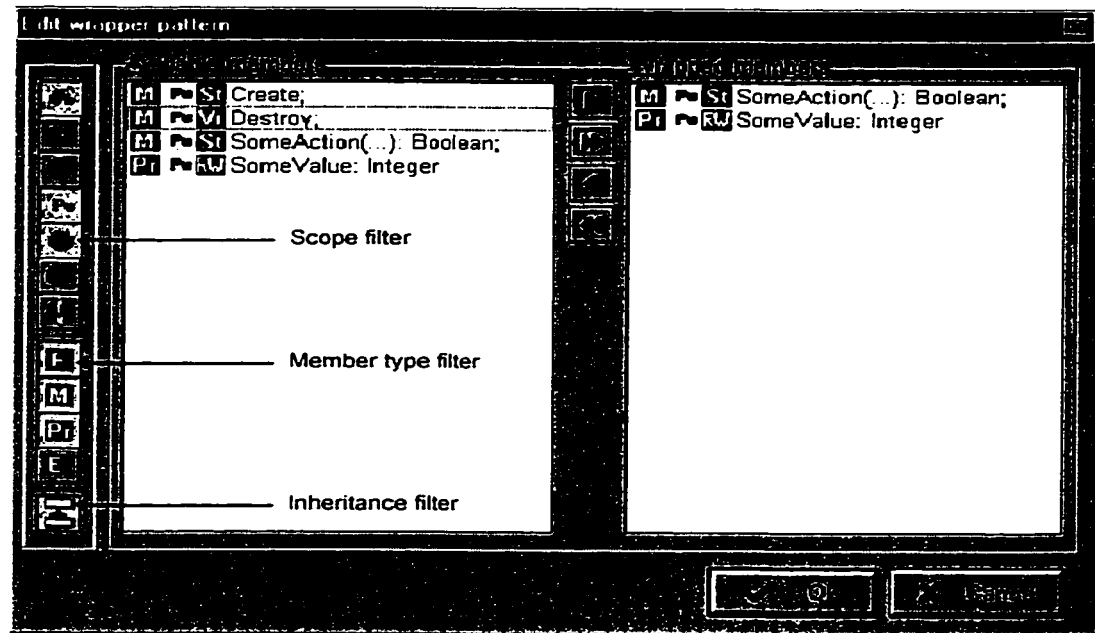


FIGURE 2.5: DESIGN PATTERNS REFERENCE MODELMAKER

2.2.3 Framework Adaptive Composition Environment (FACE)

For code generation and pattern instantiation purposes, the software development environment FACE (Framework Adaptive Composition Environment) was developed to guide the process of instantiating patterns. It starts with a primal-schema, containing the abstract classes of a pattern and their association, and then proceeds to a meta-schema for concrete classes, operations and associations [Meijler, T. D., S. Demeyer, and R. Engel, 1997]. In the FACE approach, developing an application may cover two levels:

- 1) Composing a set of objects, as these will at least initially cooperate in the run-time application.
- 2) Composing a "schema" for an application, a knowledge structure that describes the concepts (represented by so-called "class-components") and their relationships for this

application. Such a schema is similar to a "schema" in an Object-oriented/Semantic Database Management System. FACE currently does not provide other support for the software process than graphical composition.

2.2.4 Pattern-Oriented Analysis and Design (POAD)

POAD is a new Pattern-Oriented Analysis and Design approach that utilizes patterns as building blocks (components) at the design level [Yacoub S., Xue H., and Ammar H., 2000]. The approach glues the design structure of patterns at various levels of abstraction for developing pattern-oriented designs. The POAD approach produces hierarchical traceable design models that capture interaction between patterns (Figure 2.6). Currently this tool is deficient in the following features:

- This tool does not allow the user to define a pattern and save it for further reuse.
- The user has to browse through the patterns by launching the pattern browser applet outside this application. And after choosing a pattern, the user has to comeback to the application to import that pattern. Therefore, this tool is failing in the usability of pattern.
- The tool does not have a feature to save design diagrams; therefore, it could not open them in the further.
- The tool does not have any code generation capabilities [Keerthi K Arutla, 2000].

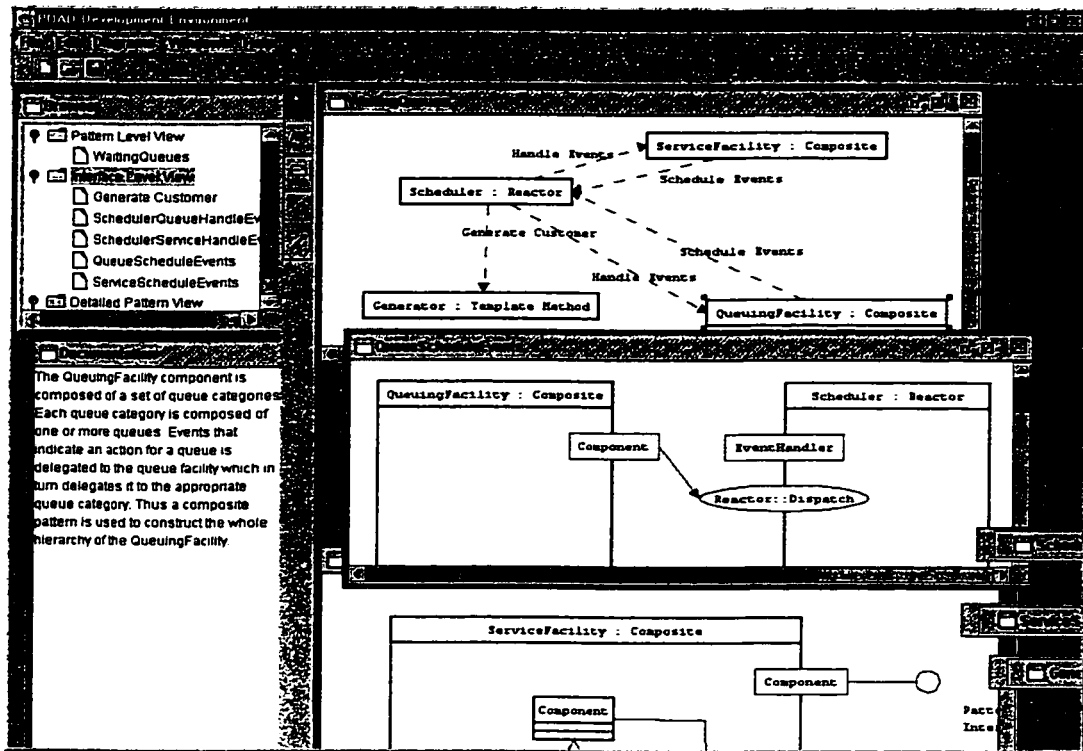


FIGURE 2.6: PATTERN-ORIENTED ANALYSIS AND DESIGN

2.2.5 OTW (Object Technology Workbench)

OTW[®] is an Object Oriented modeling tool supporting the entire software engineering process and the following operation:

- Requirements Analysis
- Modeling
- Constructing
- Deployment

OTW[®] enables the modeling of the business processes up to the software which supports them. It can be used by software developers as well as by project managers

and employees in various special departments [OTW, 2001]. The menu-driven instantiation of patterns with OTW is shown in figure 2.7.

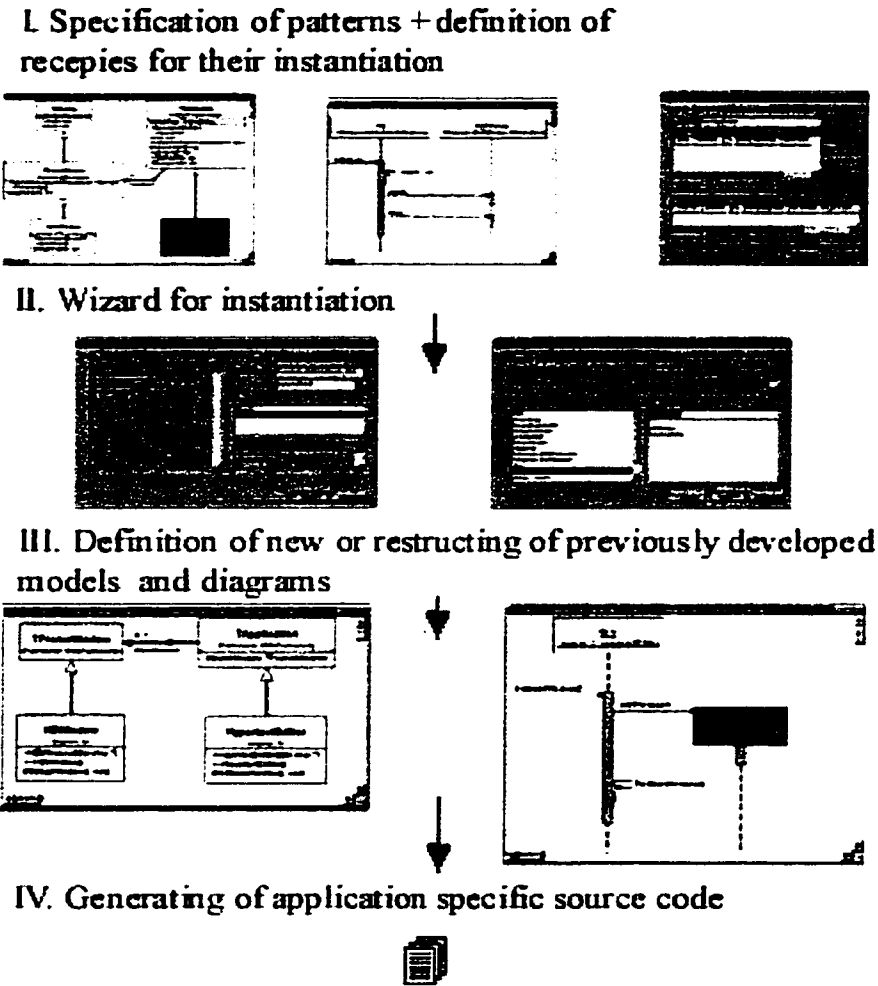


FIGURE 2.7: OTW

2.2.6 Objecteering C++ Developer

Objecteering associates Design Patterns and code generators to allow user to generate code from the dynamic model of the application [SOFTEAM, 2001]. It provides you with the Gamma State Automated Design Pattern for the automatic conversion of the UML state diagram model into a class model. The code generator then converts this class

model into C++ code. The State Design Pattern gives you a failsafe solution for generating C++ code, which corresponds to state diagrams, and guarantees a truly excellent, high-quality result.

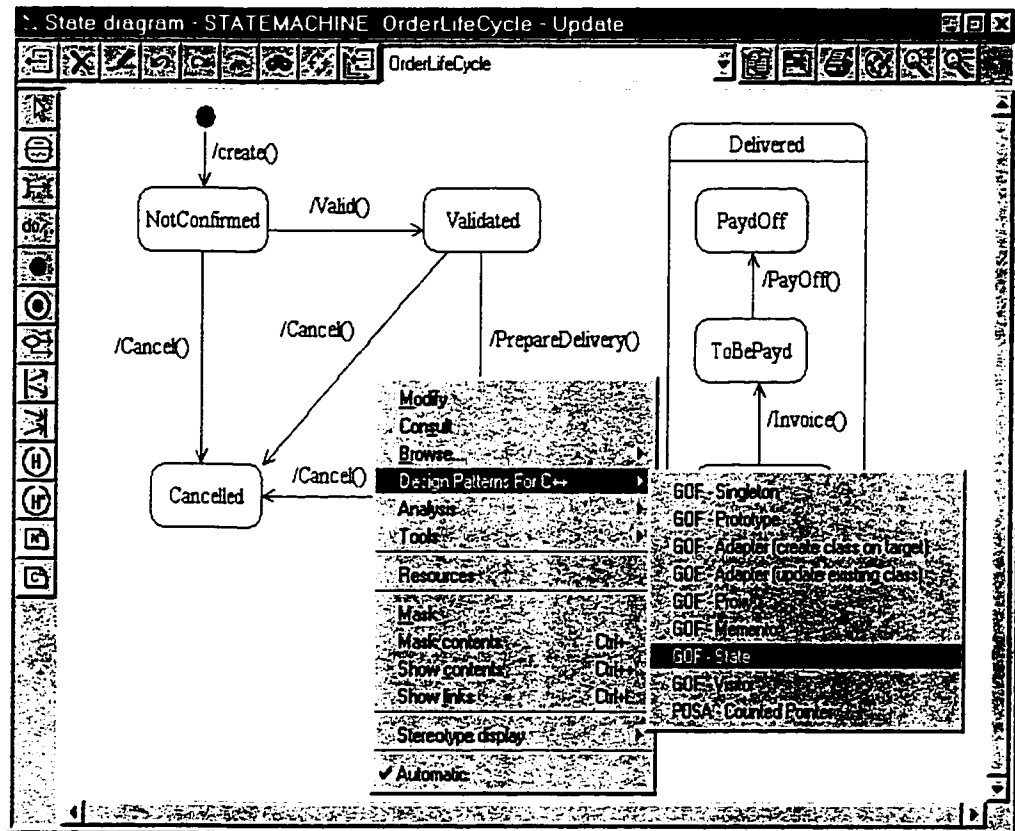


FIGURE 2.8: OBJECTEERING C++ DEVELOPER

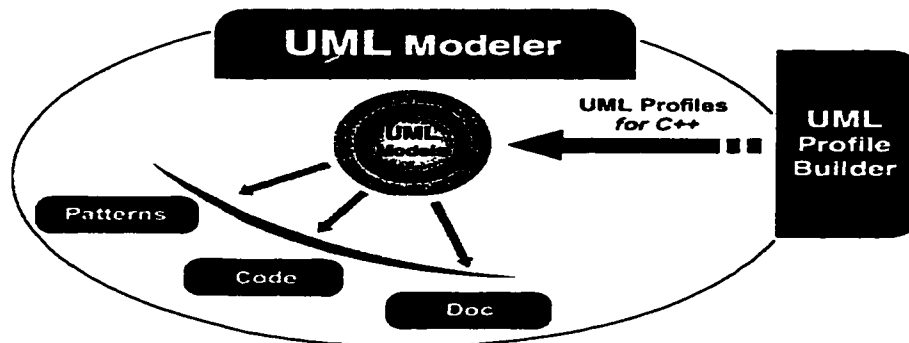


FIGURE 2.9: UML MODELER IN OBJECTEERING C++ DEVELOPER

Currently this tool is lacking in the following features:

- This tool does not allow the user to define a pattern and save it for further reuse.
- The developer cannot browse existing patterns.

2.3 Perspective: Towards an XML-Based Tool

PML (Pattern Markup Language) is an XML-based format to describe software patterns. A pattern represents a recurring solution to a software development problem within a particular context. Patterns identify the static and dynamic collaborations and interactions between software components. In general, applying patterns to complex object-oriented applications can significantly improve software quality, increase maintainability and support broad reuse of components and architectural designs [Suzuki J. and Yamamoto Y. (1998)].

2.3.1 PCML (Pattern and Component Markup Language)

ObjectVenture announces the specification development of PCML (Pattern and Component Markup Language), a break through for realizing object reuse by simplifying the use of design patterns in the software development process. PCML allows patterns at any level of abstraction to be expressed in a tangible, standard format. By leveraging the openness and flexibility of XML, this technology enables architects and developers to easily and effectively describe, package, exchange, and extend their own patterns as well as those created by others. At the same time, tool and repository providers are empowered to automate much of this process [PCML (2001)].

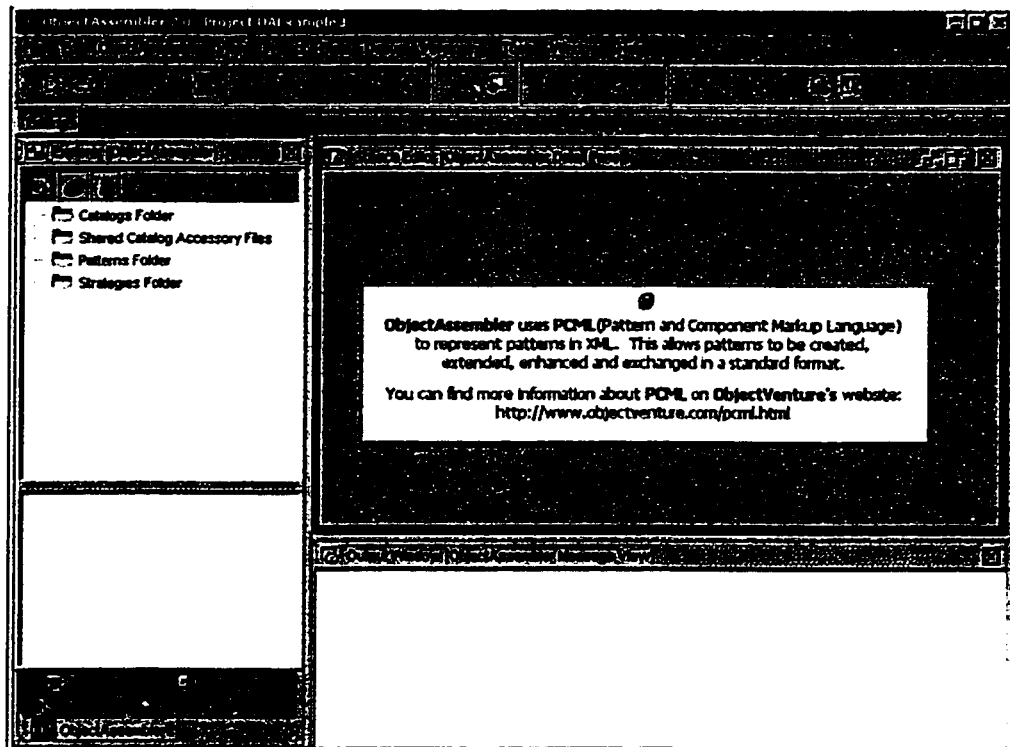


FIGURE 2.10: PCML (PATTERN AND COMPONENT MARKUP LANGUAGE)

2.4 Thesis Context

As we examine these software automation tools, we should recognize that these tools automate software implementations in the user interface development. They create specific classes and methods on the UML workspace. However, they do not automate the designer's deliberations. The designer must still analyze the forces and context and then choose which pattern to use. User interface development and modeling patterns in UML notation is a complex and high order process. The professional software designer must still understand and know how to select the appropriate options offered by wizard dialog boxes. Thus the role of the programmer as designer is not automated.

The main goal of this thesis is: to explore and share a diversity of perspectives on patterns and pattern languages for interaction design, HCI, and related design activities. We aim to identify user interface developer needs from a pattern tools and how using HCI patterns will help to reach this goal. As our investigation, we find that there are large amount of HCI patterns that user interface developer can use. However, there are not any powerful tools to give power to developer use this HCI patterns.

The important issue is “how we can use the existing software engineering tools in the Human Computer Interaction”. To enhance our understanding of the concept of HCI patterns and pattern languages in relation to other theoretical models used in software design, we need to create a tool that reduce the gap between HCI patterns and user interface developer. The second goal is to understand how computer tools can best support the various activities involved in both the creation and consumption of patterns and potential software components relating to the patterns in HCI area. We aim to identify ways in which the capabilities of the computer can enhance and support these activities.

It is also clear that existing pattern tools is not sufficient for our purposes. To achieving our goals, we try to use some concert of existing tools like POAD.

2.5 Summary

In this chapter we have described our principles research fields upon which this thesis is founded: how we can use HCI patterns and what functionality of tools are useful to achieve this goal. The aim of this is to provide a general background to existing and ongoing research in these areas. As we described in this chapter, a good pattern-oriented tools have to assist:

- A pattern developer to provide patterns more consistent in terminology, and more organized.
- A user interface developers to select and edit a categorized patterns relevant to a specific task, and to find relevant software components, configured appropriately by using knowledge embodied in the patterns.

In subsequent chapters we present our own contributions in more detail, and also present detailed analysis of our tool that will support above benefits.

Chapter 3

UPADE - Usability Patterns-Assisted Design Environment

In the previous chapter we described some of the notion of behaviour preservation and hinted at the approach that will be adopted in this thesis. In this chapter we present our approach to demonstrating behaviour preservation in detail and apply it with full rigor to a concrete our pattern engineering tools.

In section 3.1, we depict an overview of UPADE. In section 3.2, we describe the functionality of UAPDE and how we can use them. In section 3.3, we explained UPADE scenario with using an example. In section 3.4, we validation UPADE application and in section 3.5, the results of this chapter are summarized.

3.1 UPADE Overview

In recent years, there has been a substantial increase in the usability, portability and extensibility of software engineering tools. Such tools will assure the program correctness and reusability of the design and identify the developer problems. The majorities of these, however, do not interpret tools are available on a limited platforms. We analyze such deficiencies and propose an extensible architecture for a distributed software engineering

tool framework by using Java and XML technology. The resulting framework –UPADE- provides a unified interface to support the development of user interface designs and improves software production. UPADE -a prototype written in Java- aims to support HCI patterns writers and user interface developers. This tool has its foundations in the software engineering community, where several tools have been suggested for HCI and patterns-driven designs. The UPADE is a process sensitive tool that aid user interface developer to browse, to search, to edit, to create HCI patterns, to design and to combine existing usability patterns, and finally to generate code from the design. Its main user interface includes the following components, as illustrated in Figure 3.3:

- “**Browse**” provides a description of the patterns, some illustrated diagrams and several practical examples.
- “**Search**” improves efficiency of the UPADE and speed up searching of patterns.
- “**Edit**” facilitates development of new pattern and editing the existing patterns.
- “**Design**” develops a systematic approach to glue patterns, supports the integration of patterns at the high design level and automate pattern composition.
- “**Generate**” generates program elements (e.g. classes, hierarchies) for a new instance of a pattern, taken from an extensible collection of "template" patterns.

UPADE specification allows patterns at any level of abstraction to be expressed in a tangible, standard format. By leveraging the openness and flexibility of Java and XML, UPADE enables developers to easily and effectively describe, search, exchange and extend their own patterns as well as those created by others. At the same time, UPADE develop a systematic approach to glue patterns, supports the integration of patterns at the

high design level and automate pattern composition. Finally, UPADE generates program elements from an extensible collection of the code.

As a tool for automating the development of HCI designs, the UPADE embodies three key functionalities.

First, the most important functionality provided by the UPADE is the possibility given for both pattern writers and developers, using existing relationship between patterns, to define new patterns or create a design or a new pattern by combining existing usability patterns.

Secondly, in order to facilitate patterns composition, the tool supports different hierarchical, traceable, HCI design levels. In our case study, three levels are possible: Pattern Level, Design Level, and Code Level. At the pattern level, developer can see description of the patterns, search a specific pattern, create a new pattern and save the pattern into the database. At the design level, developer can glue HCI patterns to design a user interface application, replace one pattern occurrence by another, and automate pattern composition. And finally at code level, developer can see the structure of the design in terms of classes, methods, associations and inheritance relationships in a particular programming language.

Thirdly, the UPADE provides a mechanism to check and control how patterns are created or modified. By using the database information, UPADE automatically examines the patterns and offers a feedback to the designer.

This feedback received from developers conduct us to investigate XML (Extensible Markup Language) as an easy and adaptable structure to document patterns.

XML has proven its powerfulness as a markup language for documents structured information in different fields. With patterns being represented as structured information, XML can be the logical choice for not only structuring the knowledge about patterns, but also for generating the code from a high-level description of HCI design. In this way, UPADE is intended to provide a multi-platform device-independent authoring environment for accessing and manipulating usability patterns. UPADE supports the generation of a complete and operational user interface from patterns. The user interfaces that are generated may be based upon a Markup Language or a programming language. The rendering support will be provided by style sheet languages such as Cascading Style sheets (CSS) or Extensible Style Language (XSL). We have identified the potential candidates for visual -The Extensible Hyper Text Markup Language (XHTML), Scalable Vector Graphics (SVG) and Wireless Markup Language (WML) - and speech markup (VoiceXML) Figure 3.1.

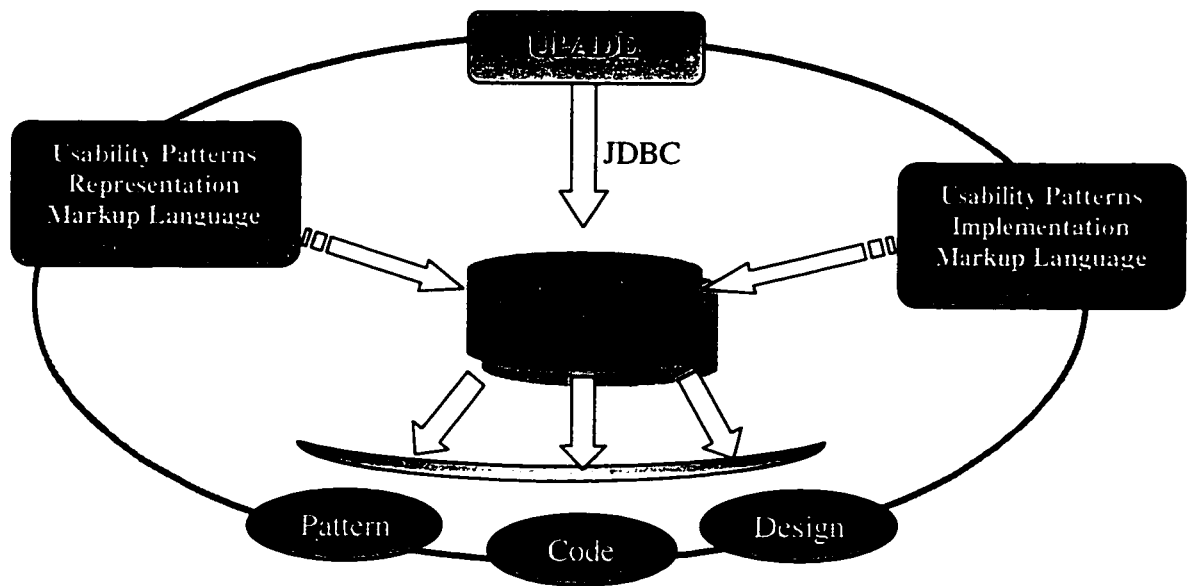


FIGURE 3.1: A SCHEMATIC OF THE UPADE FRAMEWORK ARCHITECTURE AND PROCESS

3.2 UPADE Functionality

Each pattern is defined by the attributes and associations enumerated [PCML (2001)]. Each pattern describes a collaboration of elements that provide a solution to a problem. A graphic representation of a pattern's association with other elements is provided in Figure 3.2.

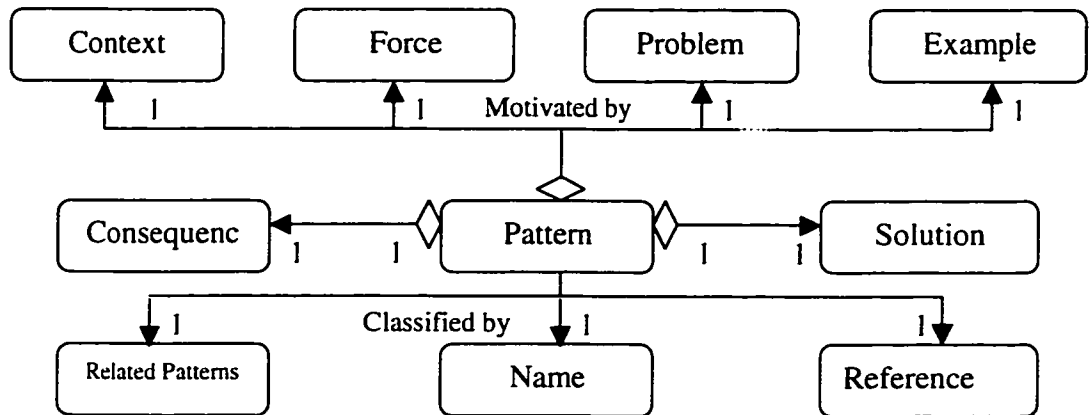


FIGURE 3.2: PATTERN ELEMENT ASSOCIATIONS

A brief explanation of the pattern elements is as follow [PCML (2001)]:

- **Process Pattern** is a collection of general techniques, actions, and/or tasks (activities) for developing a user interface development.
- **Intent** is a short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design issue or problem does it address?
- **Consequence** represents pattern usage. It describes how a pattern supports its objectives and the trade-offs in doing so.
- **Context** represents the environment within which a pattern describes itself and is a general motivation for its existence.
- **Force** represents a motivation of a pattern. It essentially amplifies the problem a pattern is trying to address and then serves as a constraint on the solution.

- **Problem** represents a design need that is to be addressed by a pattern. It essentially distinguishes the use of one pattern over another.
- **Solution** solves the problem described in a pattern. It is composed of a number of participants and defines the static structure and dynamic interactions of them.
- **Related Patterns** represents a relationship between two patterns. A pattern relationship is purely descriptive, but it does have an attribute that specifies what type of relationship it is. This element would be used to refer to a like pattern or to describe a pattern nesting.
- **Reference** represents the place or the person that create the pattern.

UPADE is designed to grant the following functionality to the software developers:

3.2.1 Browsing and searching patterns

UPADE in “Browse” mode can provide user interface developers and HCI patterns developer with just-in-time details and information on both products and process patterns. The information is depicted into unified format consisted of the pattern elements such as pattern name, context that give description of the pattern, some illustrated diagrams and several practical examples. The UPADE provides a unique repository for storing HCI patterns and these pre store HCI patterns will be available for pattern developers as demand.

When software developers run the UPADE application, UPADE will be open in ‘Browse’ panel by default, which will be similar to the one in Figure 3.3. In ‘Browse’ mode, UPADE produce and deliver patterns information, which software developers

need. The information is described into incorporated format consisted of pattern process, pattern category, pattern name, pattern description with all the relative information and examples. As you can see, the structures of **Process** and **Category** tree are presented in an **explorer** style. By default, UPADE will be browsing patterns with their Process name. However, software developers will be capable of switching to the browse by **Category** name simply by selecting the “browse by category” from **combo box** that located in top of tree panel, which is similar to the one in Figure 3.3. As you can see in Figure 3.4, when the software developers select any pattern category from the tree, the relevant patterns would be itemized instantly in a ‘Menu Bar’, which appears in top of the ‘Browse’ Panel.

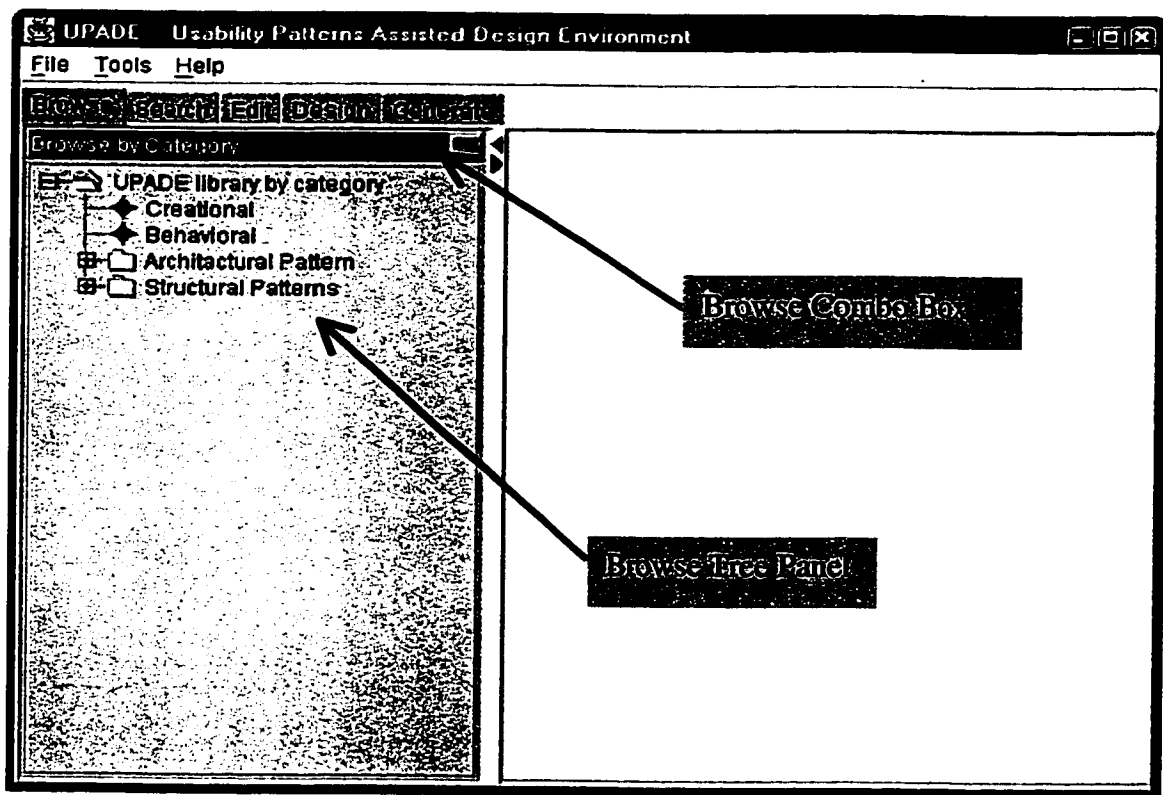


FIGURE 3.3: BROWSE PATTERNS BY CATEGORY NAME.

Once the software developer clicks on any specific pattern in the 'Menu Bar', all the relative information of that pattern will appear in the 'Text Box' area. As we can see on the following example, once the developer selects "The Chain of Responsibility" button from "Menu Bar", all information of this pattern will appear on "Text Box" area (Figure 3.5).

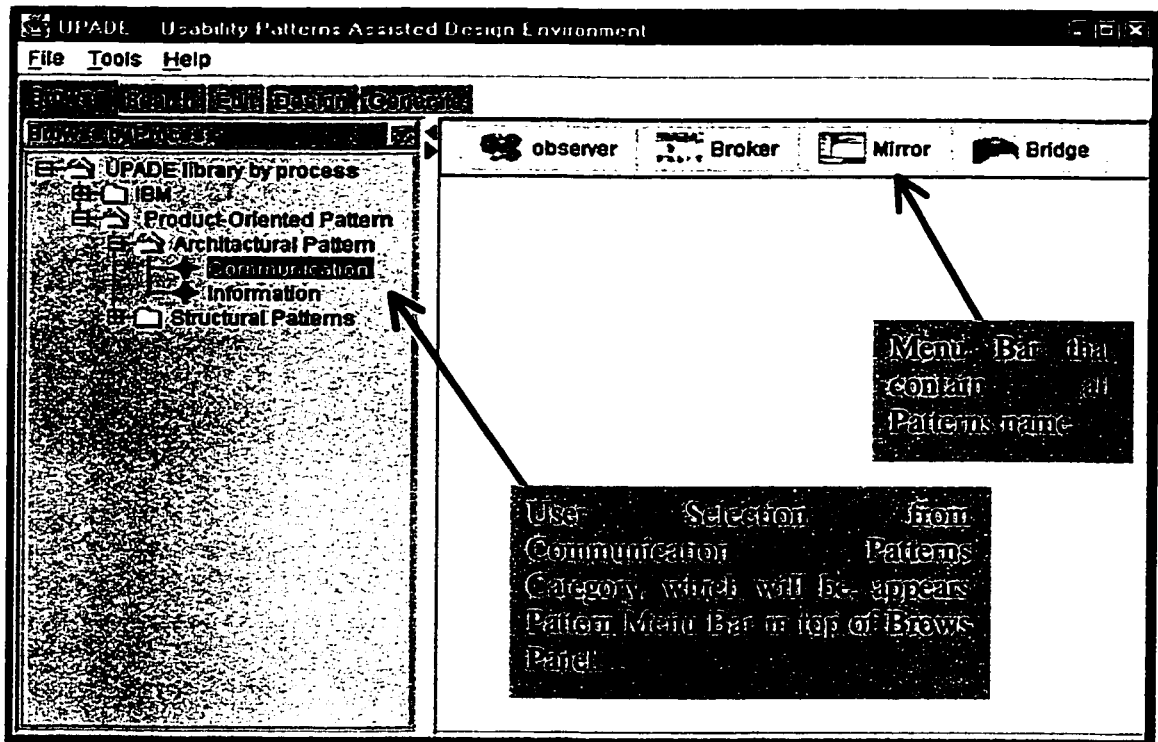


FIGURE 3.4: BROWSE MENU BAR

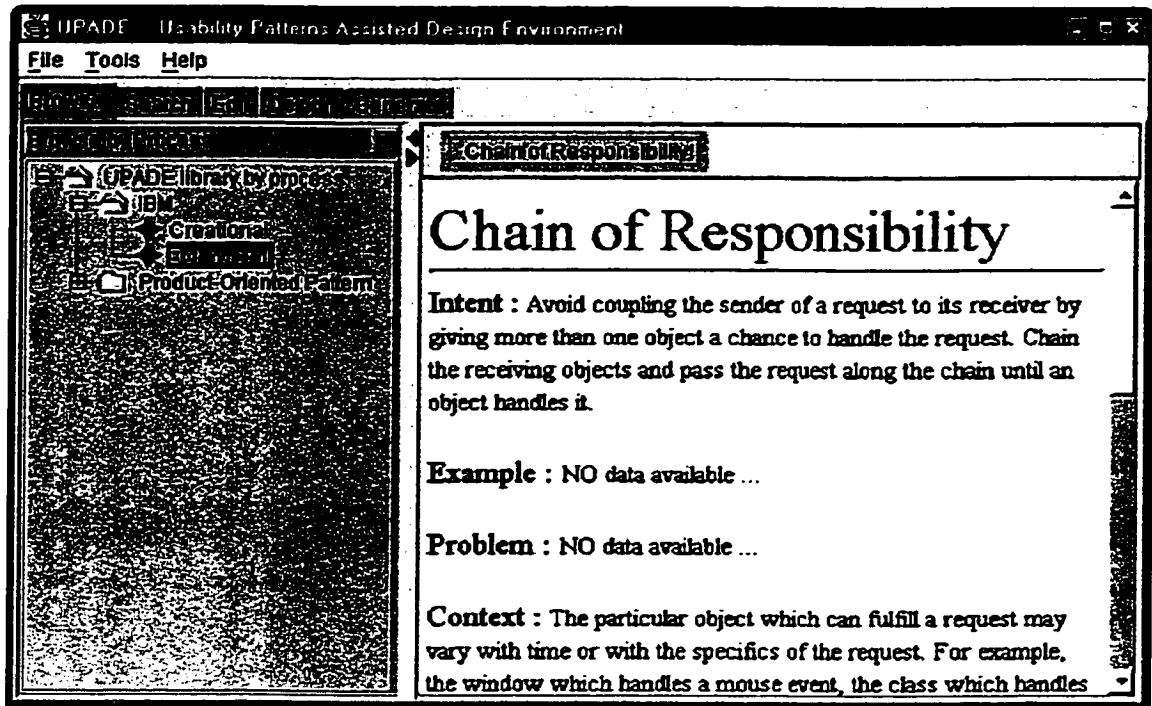


FIGURE 3.5: PATTERN DESCRIPTION

To improve the UPADE efficiency, we provide the “Search” functionality. UPADE in “Search” mode will accelerate finding new information about specific HCI pattern and will reduce the time needed to design a user interface application. The following window is the “Search” selection screen presented to the developer when developer has selected the “Search” tab (Figure 3.6). Here on the **Search Tab**, you can search for particular pattern or pattern criteria by simply typing a keyword and selecting appropriate criteria from ‘**Criteria Combo Box**’ and then clicking on ‘**Search**’ button. Let’s look more closely at the searches that Software developers are able to make.

At the simplest level, Software developer types in a word or part of a word in “**Keyword Text box**” located in top of the “**Search**” Pane and selects appropriate criteria from “**Criteria Combo Box**” which word is belong. Then, the developer clicks on the

“Search” button. UPADE search engine will search the UPADE database for the word that the developer typed in “Keyword Text box” and write back a list of all patterns in Tree format in the “Search” result Panel. For instance, if Software developer decides to search for information related to “Bridge Pattern”, in the first step, developer type “bridge” or part of pattern name into the ‘Keyword Text box’ and then by selecting “Name” from ‘Criteria Combo Box’, inform UPADE that this word is belong to the pattern name. Then, the result of search will be appearing in the “Search Result Tree” (Figure 3.6). Finally, the developer needs to select one of the results from “Search Result Tree” to see the related pattern information in the “Search Result Pane” (Figure 3.7). The Criteria that “UPADE Search” supporting is “Name”, “Solution”, “Context”, “Problem”, “Consequence” and “All” (Figure 3.8).

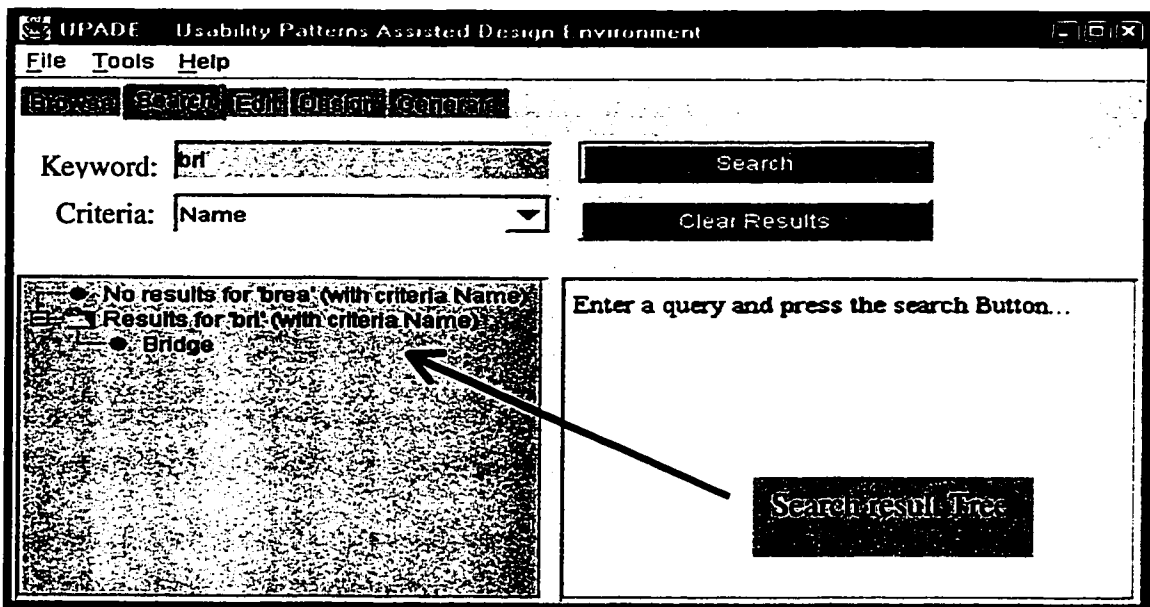


FIGURE 3.6: SEARCH RESULT TREE

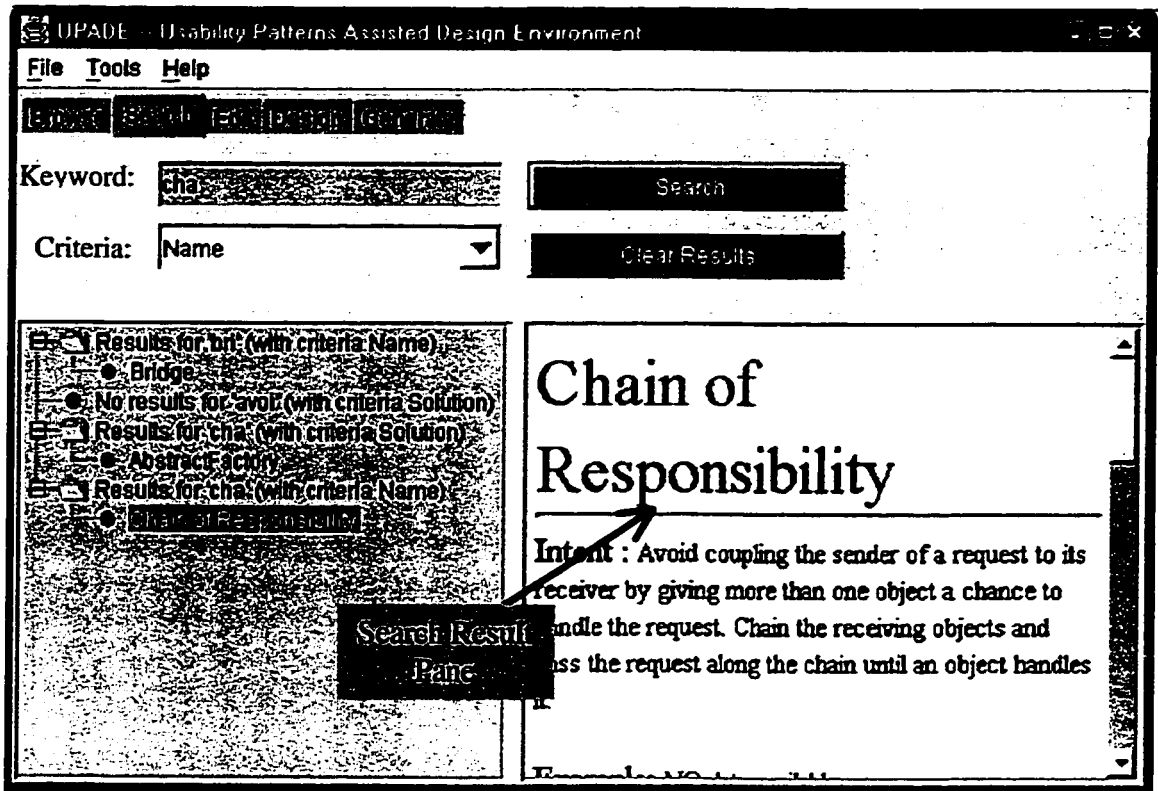


FIGURE 3.7: SEARCH RESULT PANE

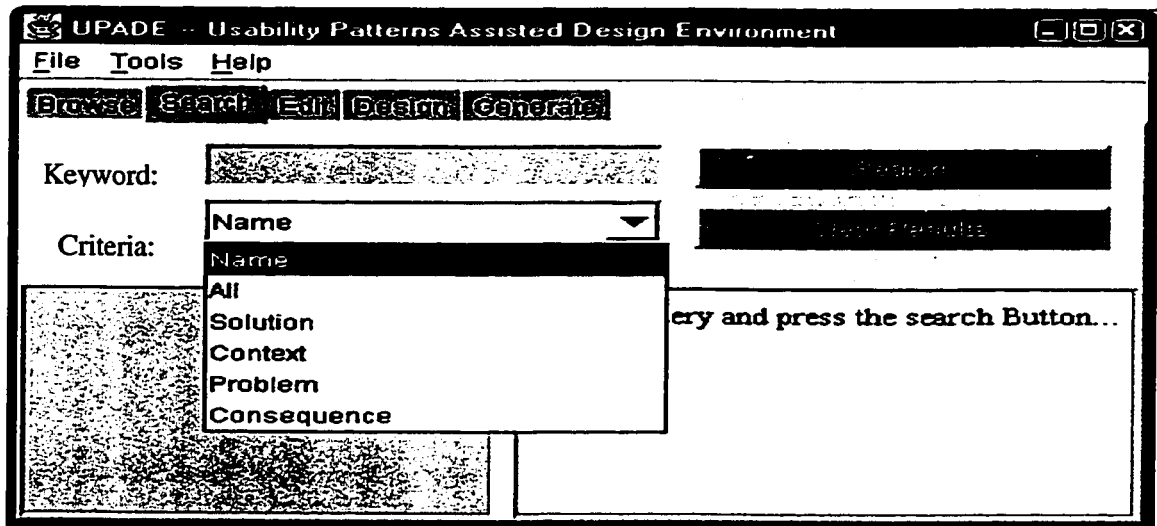


FIGURE 3.8: SEARCH CRITERIA

3.2.2 Facilitating the process of capturing patterns (Edit)

UPADE in “Edit” mode helps developers to create their own patterns or modify an existing pattern. Since patterns are reusable components, a well-developed pattern should be saved for reuse in other designs. By adding “Edit” tab into the UPADE, developers can create their own patterns and illustrate those patterns. Therefore, “Edit” will be allowing end users full control over their design standards, and the implementation patterns in code that meet those standards. The patterns applied by UPADE are sensitive to class and object characteristics. The use of characteristics allows developer to say how certain elements of a code pattern are applied to classes, objects and their parts. When a software developer selects the “Edit” tab, UPADE will open a new panel that is similar to the one in Figure 3.9.

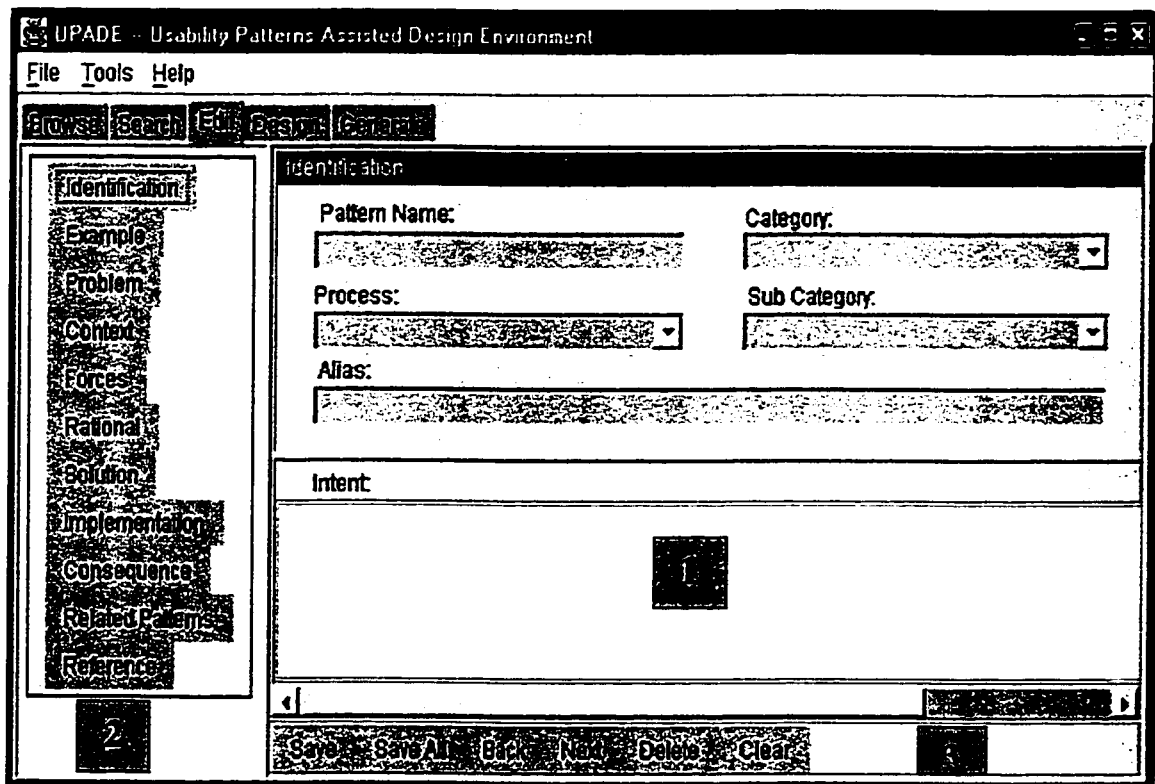


FIGURE 3.9: EDIT MAIN PANE

As we can see from Figure 3.9, the main interface of Edit pane is separated into the three areas:

- 1- **Construct Pane:** This Panel contains eleven different layouts which they are named as follow: “**Identification**”, “**Example**”, “**Problem**”, “**Context**”, “**Forces**”, “**Rational**”, “**Solution**”, “**Implementation**”, “**Consequence**”, “**Related Patterns**”, and “**Reference**”. These layouts provide a detailed specification of pattern elements. In “**Edit**” mode, UPADe will being open by default in the “**Identification**” pane that it has exposed in Figure 3.9.
- 2- “**Navigation Pane**”: This Panel contains eleven different buttons as same as Construct Pane’s layout which give power to software developer to navigate processing of pattern creation or modification.
- 3- **Control Toolbar:** The Control Toolbar is displayed at the bottom of the “**Edit**” pane under the “**Construct Pane**” Figure 3.10.



FIGURE 3.10: CONTROL TOOLBAR

It is composed of group of buttons with following functionality:

- **Save:** Saves the current layout in the “**Construct Pane**”.
- **Save all:** Saves all layouts in the “**Construct Pane**”, using the current pattern name in the “**Identification**” layout.
- **Back:** Takes you back to the previous layout pane or page in the history list.
- **Next:** Takes you forward to the next layout pane or page in the history list.
- **Delete:** will delete current pattern from the database.

- **Clear:** will clear all the text in all the panes in the “**Construct Pane**”.

3.2.2.1 Appending a New Pattern to the database

Once the “Edit” Tab selected, Software developer can start to add new patterns to the database. In UPADE, if pattern enter for the first time it is just need to follow the following steps and add your new pattern. However, if the pattern name already exists on database, UPADE will fill all information that exists for this pattern in the all “**Construct panes**” layouts.

- 1- First, pattern developer need enter the name of pattern into the “**Pattern name**” in the “**Construct Pane**” (Figure 3.11).
- 2- Then, the developer needs to select the **Process**, **Category**, and any **Sub Category** name from the related Combo Box. If the names do not exist in the combo box list, the developer simply can type into the combo box text area (Figure 3.12).
- 3- Next, by selecting “**Next**” button in the “**Control Toolbar**” or selecting appropriate button from “**Navigation Pane**”, the developer can enter pattern specific information or modify existing data (Figure 3.12).
- 4- At the end, the developer can save all information in to the UPADE database for future usage (Figure 3.12).

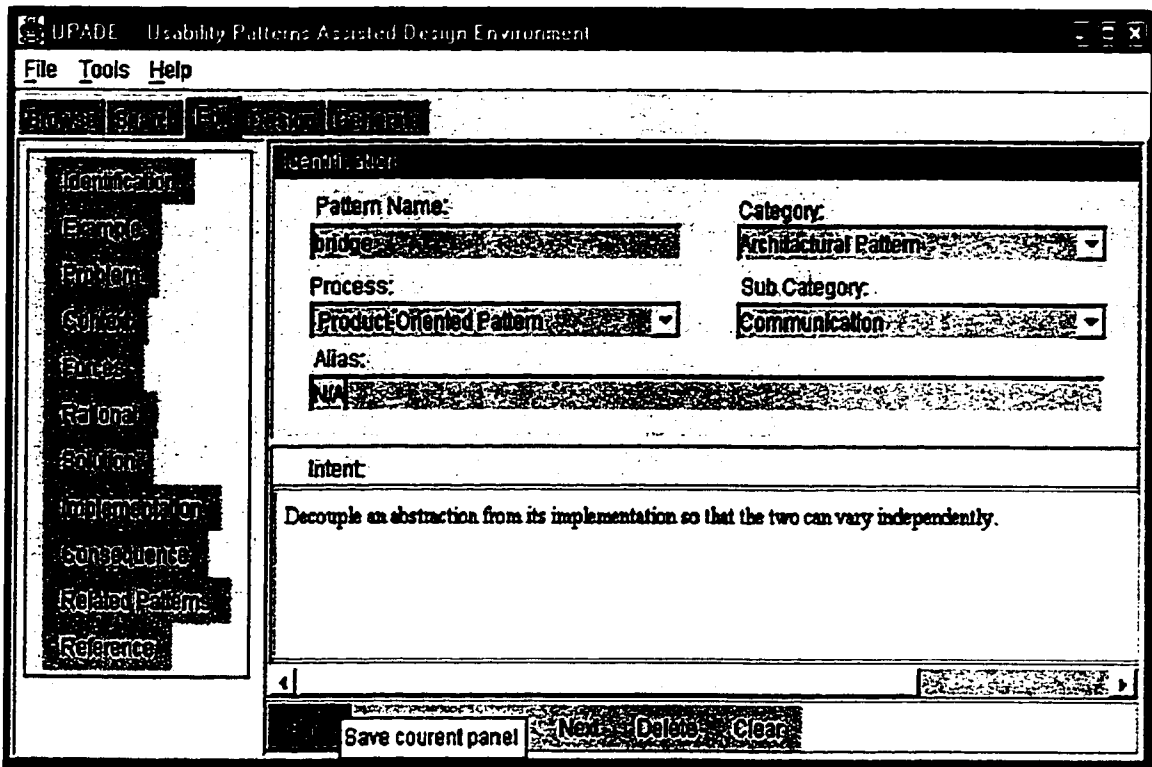


FIGURE 3.11: ENTER NEW PATTERN

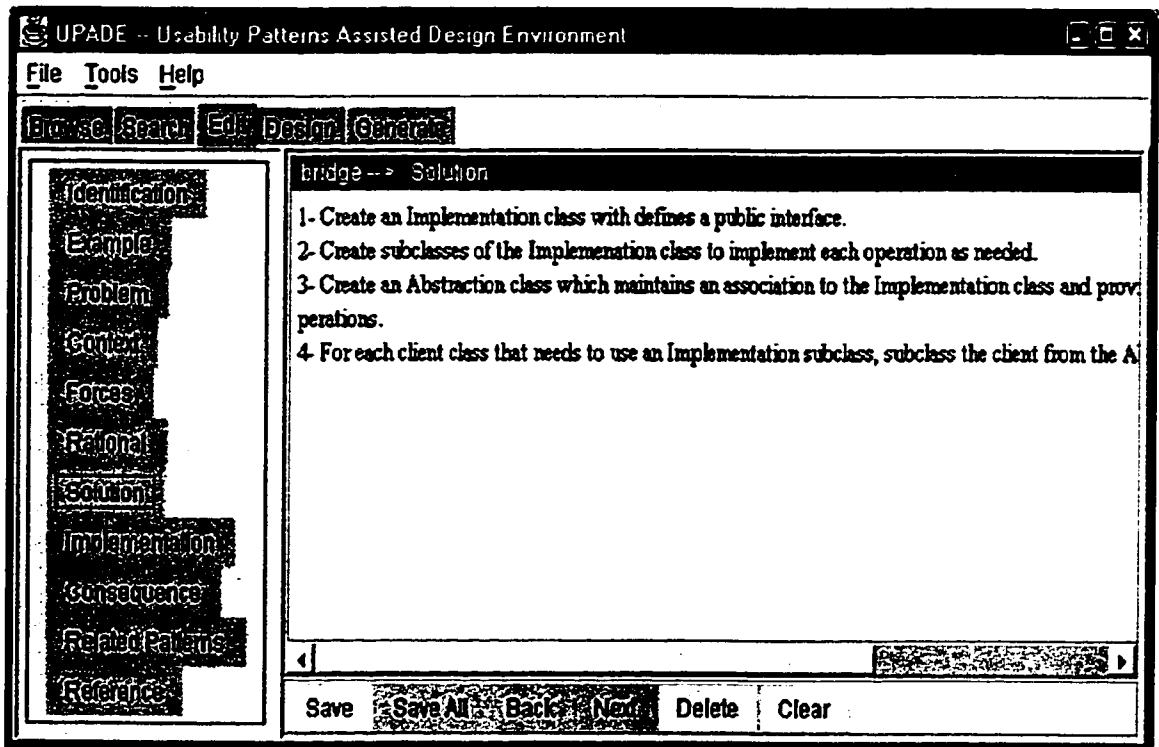


FIGURE 3.12: ENTER NEW PATTERN

3.2.3.2 UPADE Feedback

In “Edit” mode, UPADE give a different feedback to developer by pop up different Message Boxes. For example, when user traverses panels by using “Next” or “Back” button from “Control Panel” and reach to the end of panel, UPADE will pop up the appropriate Message Boxes. If developer decides to save or deletes a pattern to or from UPADE database, UPADE also have a related feedback. In the “Edit” mode, when developer traverses in the different panel layout, UPADE shows the “Path” in the top of each layout Panel (Figure 3.13) which gives enough information to the developer that this layout belongs to which pattern.

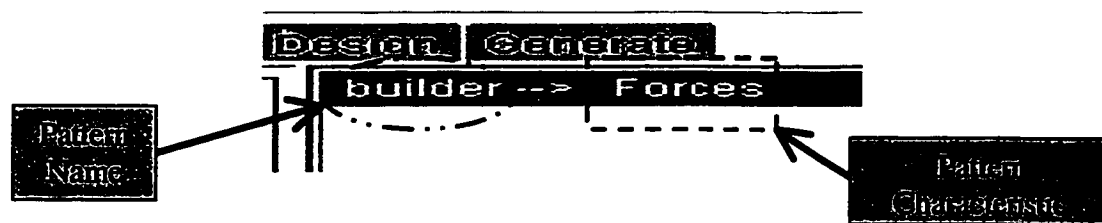


FIGURE 3.13: EDIT PATH

3.2.3 Support developers in creating complex designs

UPADE in “Design” mode help developer to glue HCI patterns and design a user interface application, replace one pattern occurrence by another, and automate pattern composition. As we know, a pattern is a some what generic description of a solution provided to address one or a common set of problems in a certain context. Although a

pattern describes a solution, it does not put any constraints on how that solution may be realized. A pattern may; however, describe how it relates to other patterns and even how it may be composed of other patterns. In this way, the abstract nature of patterns is preserved while the realization of solutions and idioms is reserved for strategies. For this reasons, UPADE gives an environment to glue HCI patterns together. Firstly in “Design” mode, UPADE can support the designing activities from more general to specific. Secondly, “Design” supports combination and organization of existing patterns. For example, The Software developer can glue Page Managers Patterns into the other Information Architectural Patterns like Navigation Support Patterns and Information Containers Patterns. Moreover, the designer has the freedom to organize Navigation Support Patterns and Information Containers Patterns inside the layout, the developer can move these patterns in the design phase and even developer can delete the useless ones from the design. The purposes of these activities aim to explore how to organize and combine the existing patterns to customize and format the new patterns. Finally, “Design” editor provides a mechanism to check the usage of patterns. It can automatically examine the compatibility of certain patterns and give the related instruction to the designer consequently. When software developers select “Design” tab, **UPADE** will open a new panel that is similar to the one in Figure 3.20. As we can see from Figure 3.14, the main interface of “Design” pane is separated into the three areas: **Browse Tree Panel, Control Menu Bar, and Drawing Pane.**

- **“Browse Tree Panel”** is the section that developer can browse the entire pattern existing in the **UPADE** database.

- “Control Menu Bar” is the place that the developer can navigate the drawing process.
- “Drawing Pane” is area that the developer can glue the HCI pattern by simply using drag and drop mechanism.

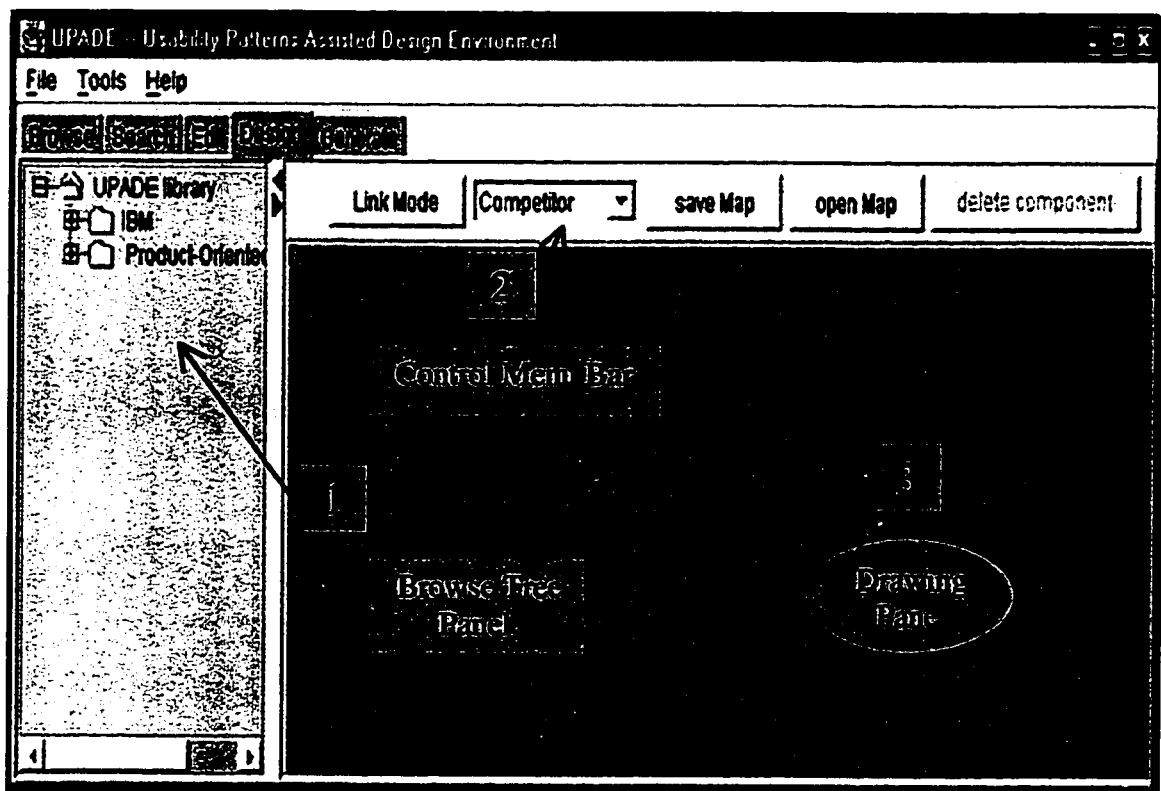


FIGURE 3.14: DESIGN PANE MAIN PAGE

3.2.3.1 UPADE Pattern Composition

Once the “Design” Tab select, developer can start to compose patterns with the following order:

- 1- First, the pattern developer needs to browse the tree in the “**Browse Tree Panel**” to find pattern name (Figure 3.15).
- 2- Then, developer needs to select the pattern name and drag and drop it into the “**Drawing Pane**” area (Figure 3.15) and repeat steps 1 and 2 till all necessary patterns put into the drawing area.
- 3- Next, by selecting “**Link Mode**” button from “**Control Menu Bar**”, the developer can connect each pattern to the other by choosing appropriate connectors that are in the Combo Box located in the Control Menu box (Figure 3.15).
- 4- At the end, the developer can save the pattern composition map into the XML format for the usage in code generation mode. When developer select “**Save Map**” button from “**Control Menu Bar**”, a new dialog box will be pop up (figure 3.16) to ask developer entering a name for pattern composition map.

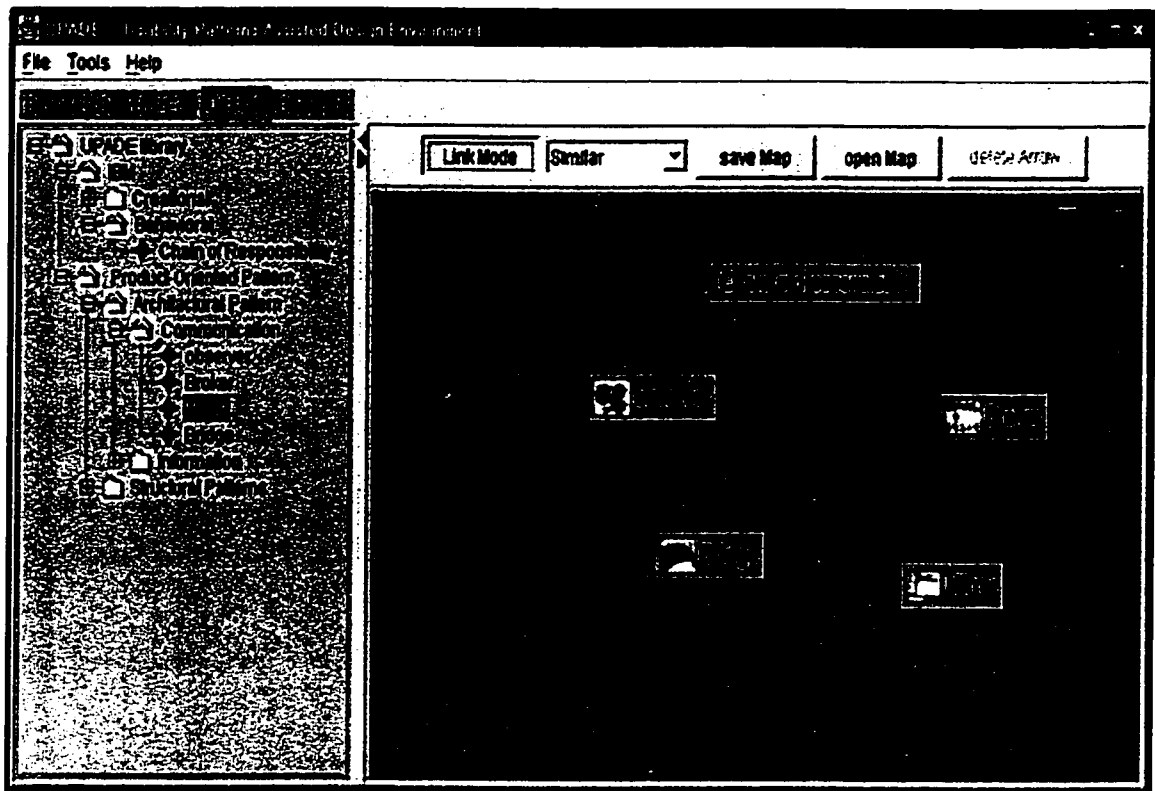


FIGURE 3.15: COMPOSING THE PATTERNS

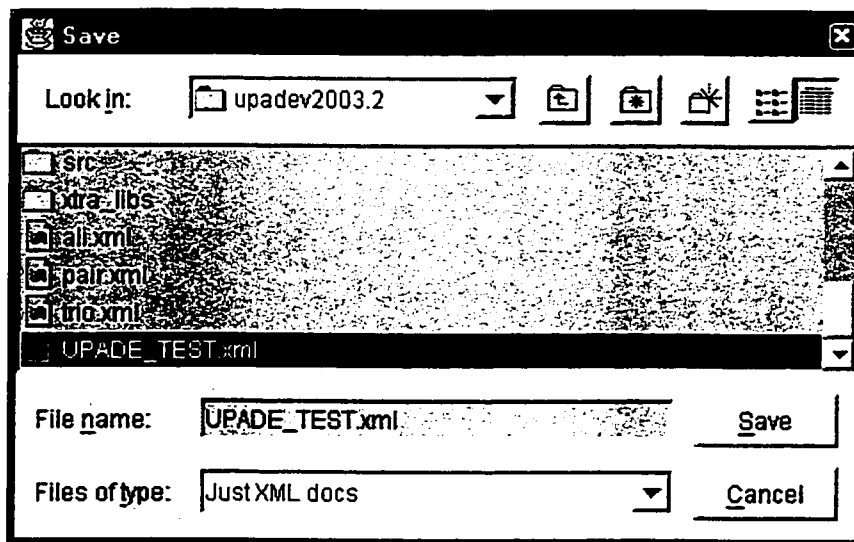


FIGURE 3.16: SAVE MAP DIALOG BOX

3.2.3.2 Modifying a Pattern Composition

First, pattern developer needs to open an existing pattern composition map or work in currently opens Map. When developer select “**Open Map**” button from “**Control Menu Bar**”, a new dialog box will pop up (figure 3.17). The developer needs to select from the file list.

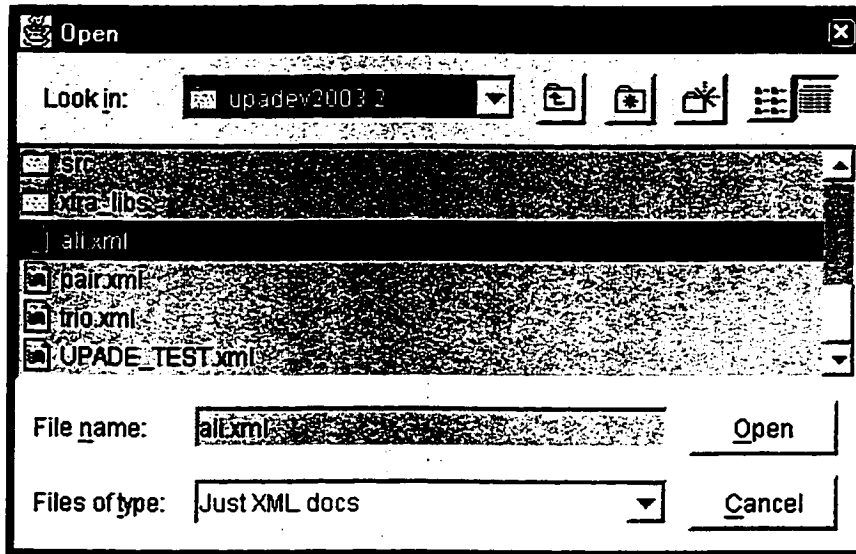


FIGURE 3.17: OPEN MAP DIALOG BOX

When the proper Map files opened, the developer is able to change any relation between patterns, delete any pattern or add new pattern to the Pattern Composition by clicking them. After modification, the developer can save it again.

3.2.4 Generating the code from a high level XML descriptions

UPADE in “Generate” mode aims developer to generate program elements like classes, hierarchies which this elements taken from an extensible collection of “template” HCI patterns code. Please note that this part of UPADE hasn’t been implemented completely in the current prototype. It will be realized in next version of UPADE. We saw that the current code generation solutions of the day did not support a development organization's need to build their own design pattern and combine implementation rules with those patterns. UPADE was created to do this directly. UPADE allows Software developer full control over their own design principles to implements a user interface code that meets those principles. UPADE is a tool that allows developer to create rules for how code is created from patterns sample code. The patterns applied by UPADE are sensitive to class and object characteristics. The use of characteristics allows developers to say how certain elements of a code pattern are applied to their classes, objects and parts. The main problem of the “Generate” is how automatically and faultlessly applies the policies and rules of implementation. This rules need to be put into effect over the entire sets of classes and objects. These types of rules can be competently expressed and automated by using XML capabilities. We hope, the developer will be intrigued by the power of basing implementation strategies around the pattern code. In this phase, operational behavior of the”Generate” is like a translator (Figure 3.18). First, it takes in a very compact specification of classes and objects (an object model), and code templates.

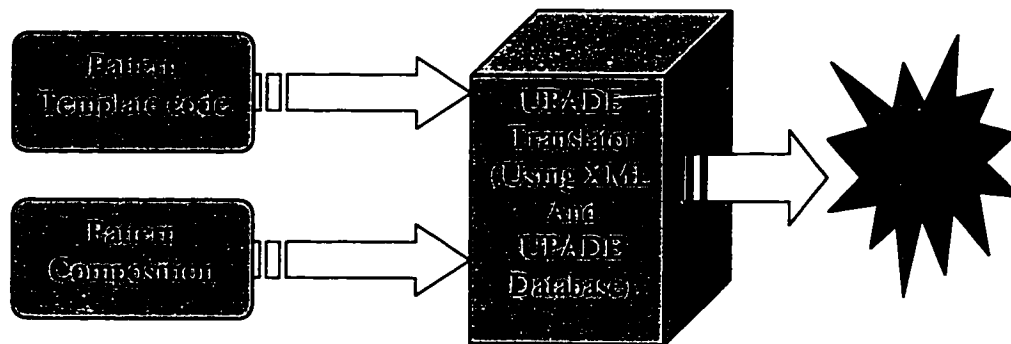


FIGURE 3.18: GENERATE TRANSLATOR

Next, it follows the rules expressed in the code templates that UPADE provide them. UPADE can generate code files and their contents by using the XML Language. The object model syntax is easy to create, read and understand. The rich capabilities of the template statements allow developer to express coding decisions in advance that take the object characteristics into the account. UPADE will allow developer to take the code in the pattern, and express them in to a template. Therefore, developer can create code by using the pattern faster and more consistently.

3.3 UPADE Scenario

Since user interface developer want to work on different levels of abstraction, UPADE provide three mutually consistent levels:

- **The Pattern Level:** where developer can see description of the patterns, search a specific pattern, create a new pattern and save the pattern into the database. For these reasons, UPADE restrain *Browse*, *Search*, and *Edit* tabs (Figure 3.19).

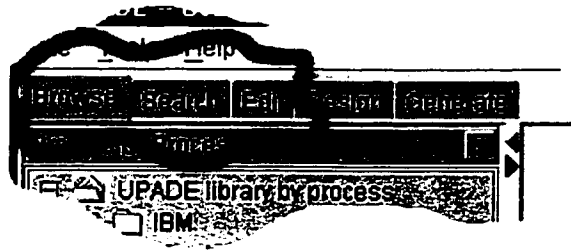


FIGURE 3.19: THE PATTERN LEVEL

- **The Design Level:** where developer can develop a systematic approach to glue patterns, supports the integration of patterns at the high design level, replaces one pattern occurrence by another one and automates pattern composition. For these reasons, UPADE contain *Design* tab (Figure 3.20).

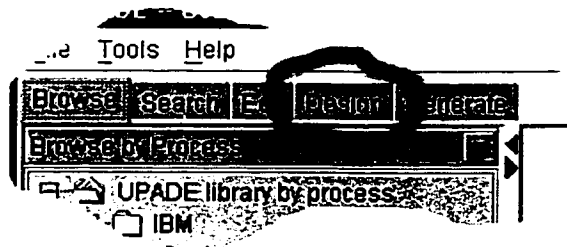


FIGURE 3.20: THE DESIGN LEVEL

- **The Code Level:** where developer can see the structure of the design in terms of classes, methods, associations and inheritance relationships in a particular programming language. For these reasons, UPADE includes *Generate* tab (Figure 3.21).

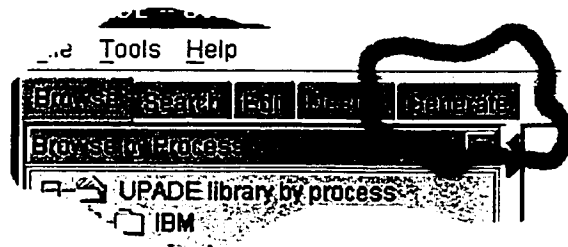


FIGURE 3.21: THE CODE LEVEL

It is better to exploit detailed examples to depict UPADE functionality by using the design level. For instance, user interface developer decides to create a web application. To create a web application first we need to define web site architecture that it is base of the collected information from the user analysis. Then, create a flow diagram that the developer can identify all web pages within the web site and show the pathways linking to each page. To create such a web site, we can use “Information Architecture patterns” such as Sequential, Hierarchical, Grid, and Composite Patterns. A complex and large web application is generally organized using a combination of several architectural patterns figure 3.22.

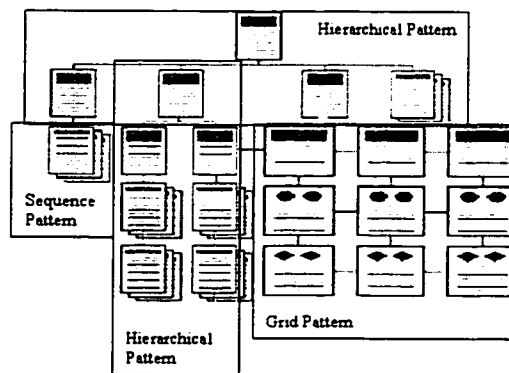


Figure 3.22: A Composite Pattern combining Sequences Pattern, Hierarchical Pattern and Grid Pattern.

To establish a prototype of this web page, user interface developer needs to go through the following steps (Figure 23):

- Select “Design” tap from the UPADE main interface.

- Browse down the “Browse Tree Panel” to find appropriate patterns. In our example, the user wants to create a home page by using homepage pattern, convenient pattern, path pattern, toolbar pattern, shortcut pattern, and browsing index pattern and some other informational pattern like Sequential, Hierarchical, and Composite.
- In the product_oriented pattern, select Architectural Patterns and then select information.
- In information node, select and drag and drop the Composite in the drawing pane.
- Continue previous step until no more pattern are needed.
- Press link mode button from Control Menu Bar and choose correct relations from the combo box that are located in the Control Menu Bar.
- Connect the entire pattern with appropriate relation.
- Select save button to save the pattern composition diagram in the XML format.

Note that if the user wants to modify an existing diagram they can follow section 3.2.3.1 and 3.2.3.2 of this chapter.

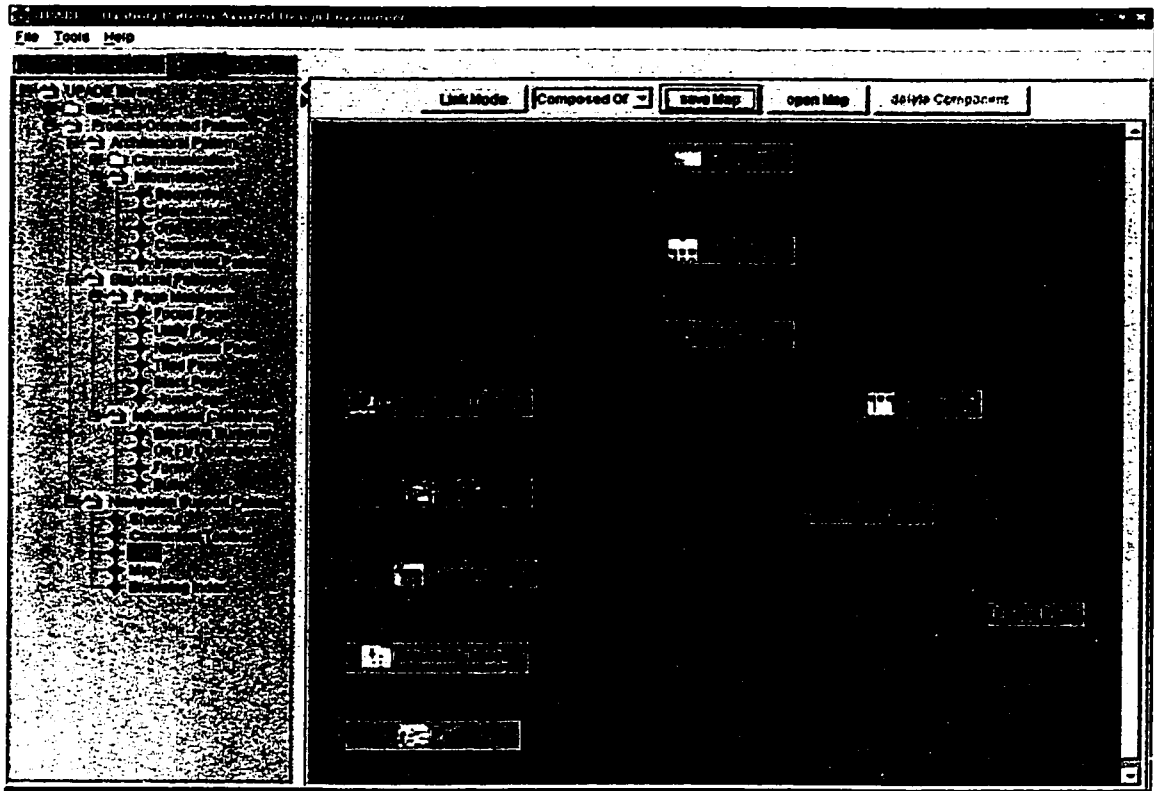


Figure 3.23: Web site creation by using architectural patterns

3.4 Validation of UPADE

One of the key differences between UPADE and other user interface development tools (such as UML tools) is that developers are in control. Developers are able to define the objects and their characteristics. Developers also are able to define their patterns, modify them and use them to create the code based on pattern characteristics. UPADE was designed to be customized and extended, with the realization that most people have achieved a local set of conventions for style and structure, and only need a tool to assist them in creating code more quickly that honors those conventions.

3.5 Summary

In this chapter we described our UPADE HCI pattern engineering tool. The tool helps user interface developer to browse and search the existing HCI patterns. Developer can also edit and create new HCI pattern by combining existing HCI patterns. In the next chapter, we will discuss the UPADE implementation issues.

Chapter 4

UPADE Implementation

In this chapter, we describe in detail the implantation of UPADE. The UPADE Framework Architecture is presented in section 4.1 followed by a brief overview of the JDBC database in section 4.2. In section 4.3 we will point out the benefit of using Extended Markup Language in pattern engineering tools. In section 4.4 a summery of the topic in chapter 4.

4.1 UPADE Framework Architecture

UPADE is an ongoing research project instructed by Dr. Ahmed Seffah. In this research, we are examining the original solution of employing usable HCI patterns as a framework for integrating usability in CASE tools while increasing the usability of systems. A schematic of the UPADE framework architecture and process are shown in Figure 4.1.

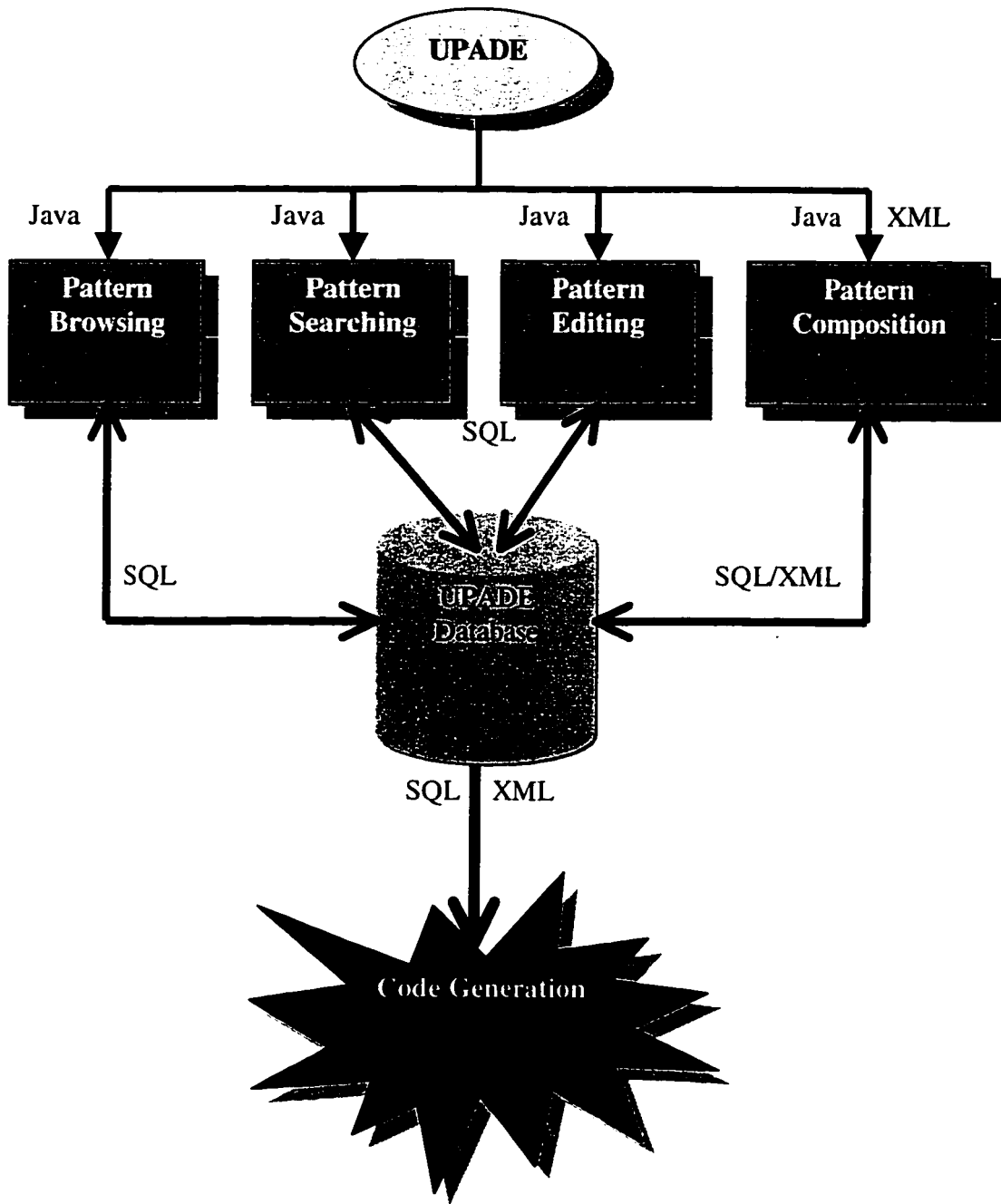


FIGURE 4.1: A SCHEMATIC OF THE CURRENT UPAGE FRAMEWORK ARCHITECTURE

The component level view of the UPADE tool is shown in Figure 3.32.

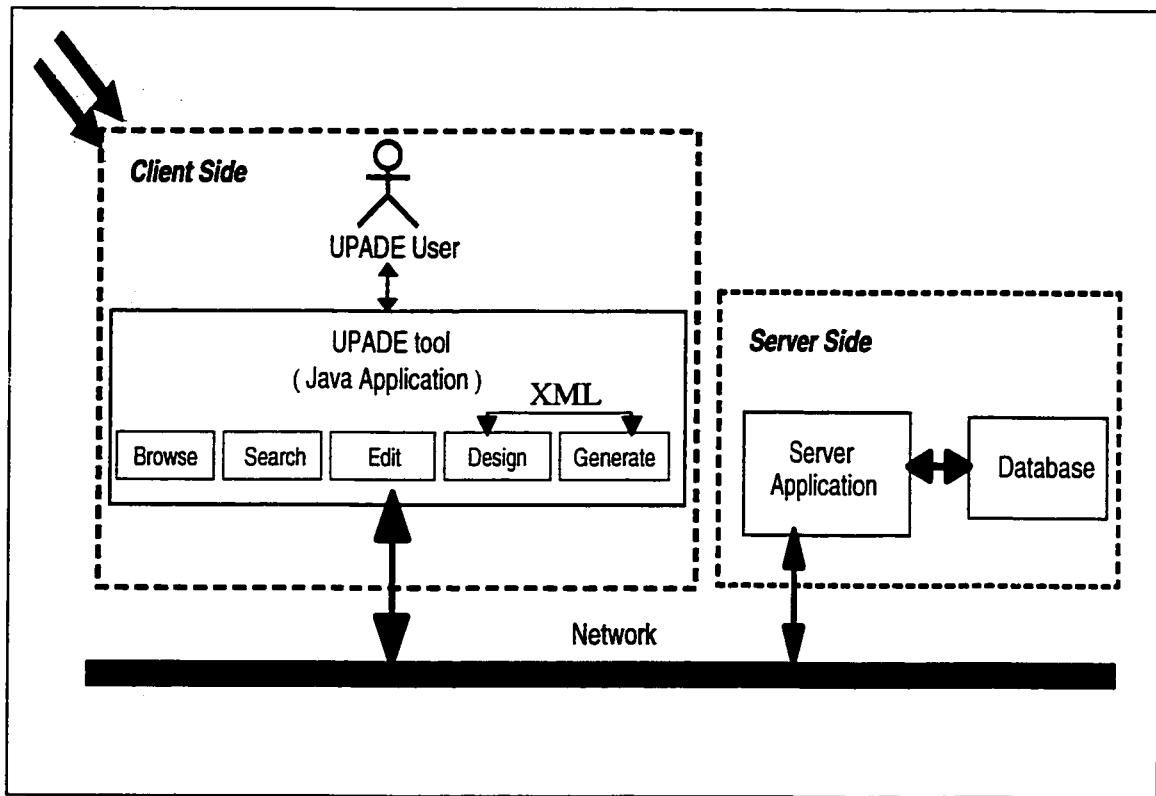


FIGURE 4.2: UPADE USE-CASE DIAGRAM

As we can see in Figure 4.1, UPADE has been implemented using Java, the JDBC and XML. The UPADE allows developers to take advantage of the Java platform's "Write Once, Run Anywhere" capabilities for industrial strength, cross-platform application. Java Database Connectivity (JDBC) technology is an API that lets developer access virtually any tabular data source from the Java programming language. It provides cross-DBMS (Database Manager System) connectivity to a wide range of SQL databases. A view of the UPADE database system is shown in the figure 4.3.

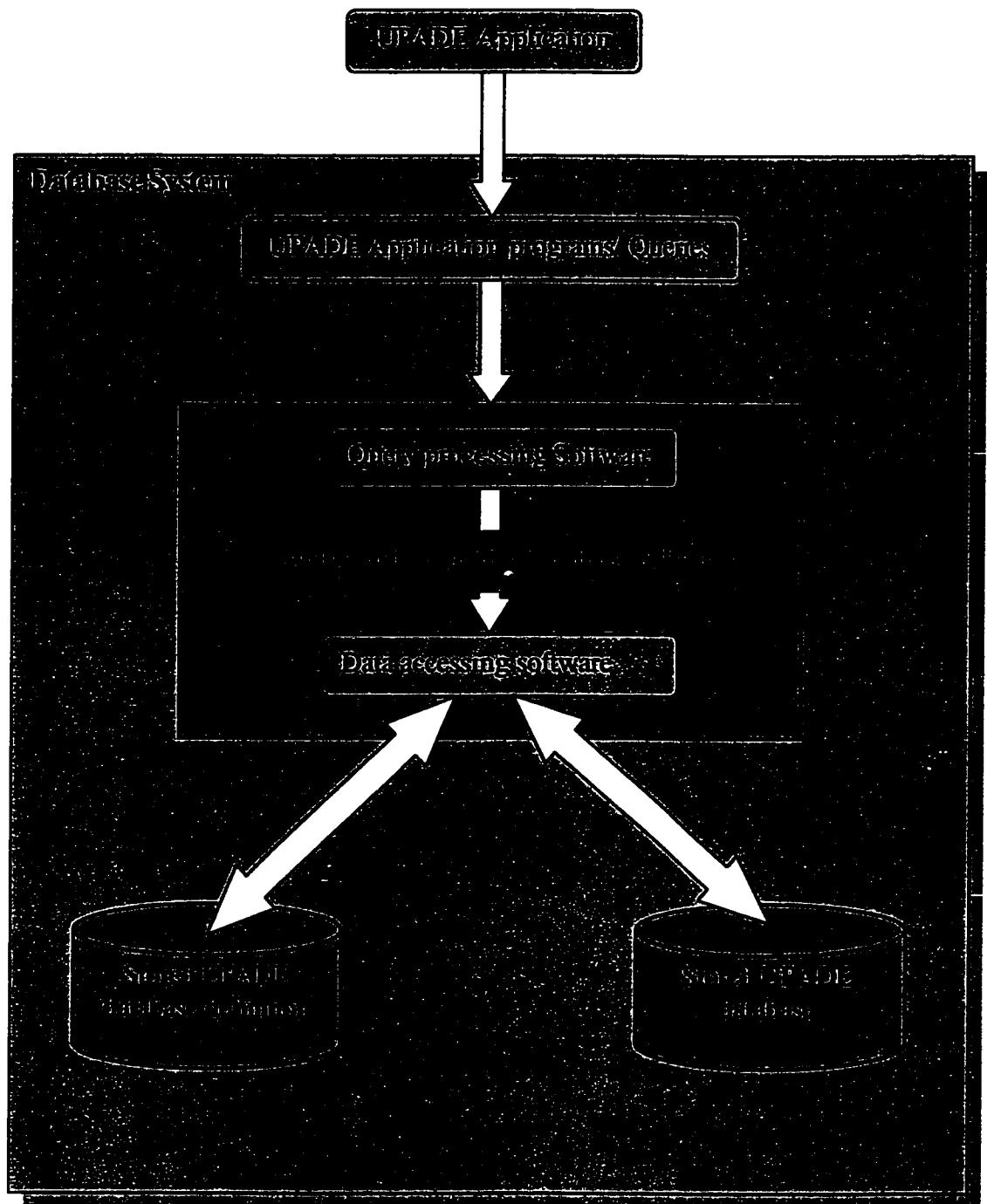


FIGURE 4.3: UPADE DATABASE SYSTEM

4.2 JDBC

Java Database Connectivity (JDBC) provides Java developers with a standard API that is used to access databases, regardless of the driver and database product. With a JDBC technology-enabled driver, a developer can easily connect all corporate data even in a heterogeneous environment. JDBC requires at least JDK 1.1, a database, and a third party JDBC driver. JDBC presents a uniform interface to databases. If a database is changed, the application only needs to change its driver. There are plenty of JDBC drivers available now that support popular databases. A suitable JDBC consists of two layers [Cay S. Horstmann, Gary Cornell (1999)]; the top layer is JDBC API. The Java application communicates with the layer below it, the JDBC driver manager API, by sending various SQL commands. The JDBC driver manager API communicates with various third party drivers, which actually connect to the database and execute the SQL commands. This gives the tool the flexibility to change the remote database and the drivers at any time, without making any changes to the existing code Figure 4.4. Right now, UPADE uses the JDataStore as its remote database and JDBC driver to connect to this remote database. In future, if we decide to change UPADE database to another database like Oracle database, all we have to do is buy a JDBC driver for Oracle from any third party and tell our application to use this new driver to connect to the database. Figure 4.5 shows the current UPADE database structure by using JDataStore.

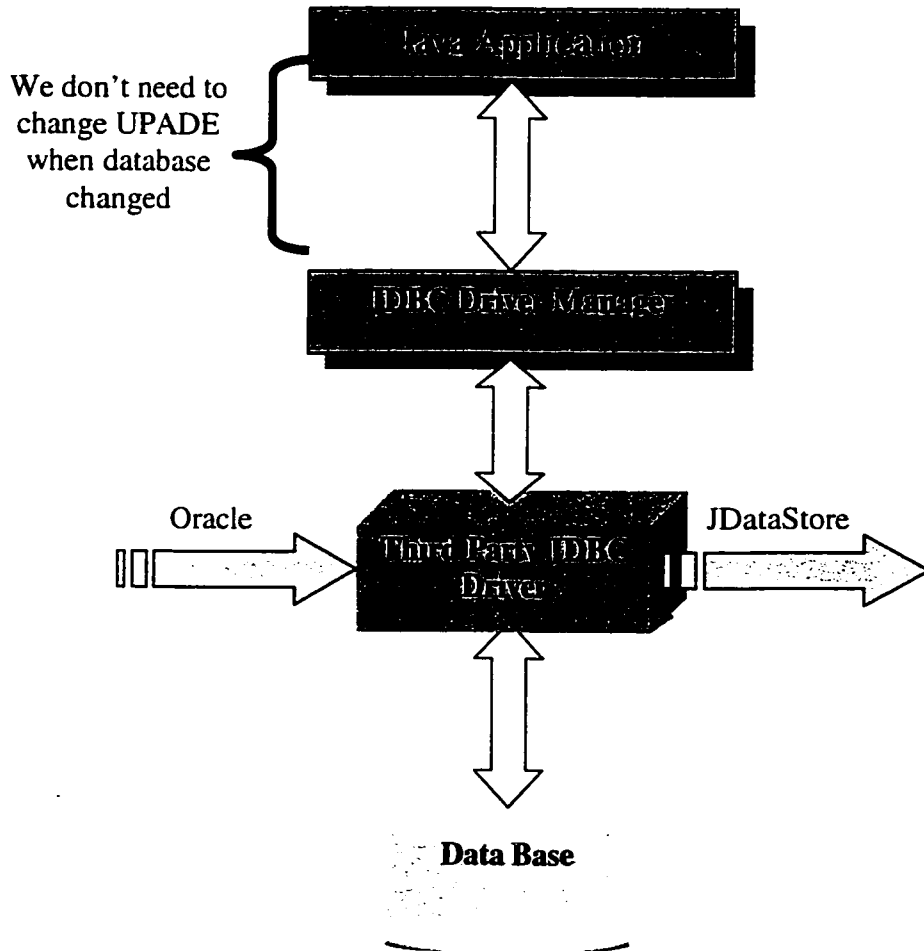


FIGURE 4.4: JDBC DIAGRAM

UPADE Database Systems Structure

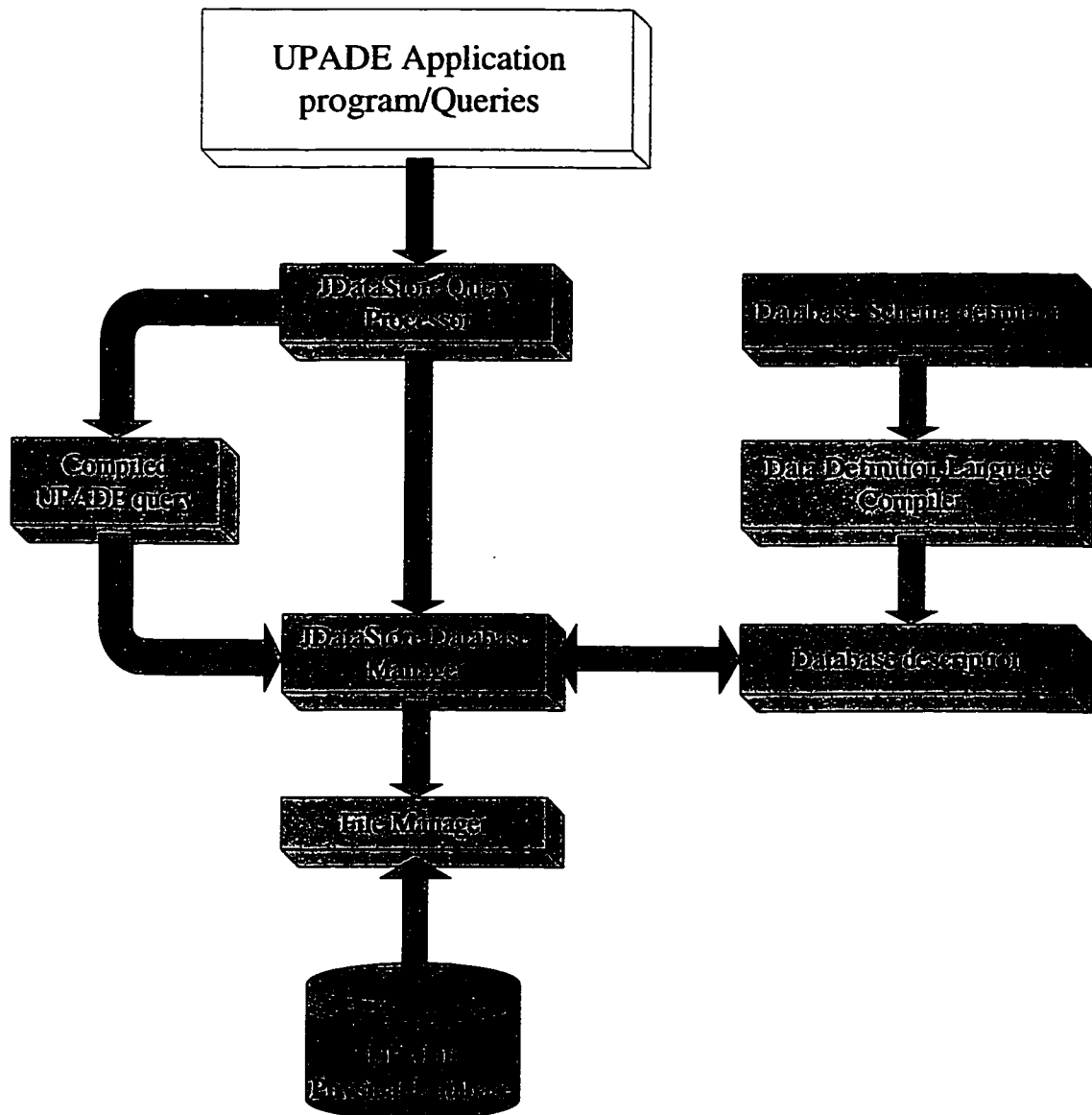


FIGURE 4.5: UPADE DATABASE STRUCTURE

4.3 Extended Markup Language (XML)

As we discussed above, UPADE can compose patterns and use them in the code generation. XML (Extended Markup Language) is the new technology, which allows patterns at any level of abstraction to be expressed in a tangible, standard format. By leveraging the openness and flexibility of XML, this technology enables architects and developers to easily and effectively describe, package, exchange and extend their own patterns as well as those created by others. At the same time, tool and repository providers are empowered to automate much of this process. This makes our user interface application more efficient and productive and programmer does not have to worry about checking the validity of the document. And also there are a lot of XML document parsers available in the market, which parse the document and gives the content of the document. This saves a lot of time for the programmer. In the following, we summarize our investigations regarding XML as the underlying structure for manipulating patterns and for automating the development of user interface designs.

4.3.1 XML as a Neutral Language for Documenting Patterns and Describing the Related Design Knowledge

Using XML for documenting patterns has already been explored. For example, the Amsterdam pattern catalog is documented using an XML. The following table describes the DTD (Data Type Definition) we defined and used for sharing via the Web our pattern language with IBIS designers Figure 4.6. Once used by the developers we interviewed, however, it became quite clear to us that this DTD, like those proposed by others, forced all pattern writers and users to closely adhere to and master

a specific format and terminology for documenting patterns. This constitutes another key obstacle towards a flexible, pattern-driven design approach. With a view to offering maximum flexibility with regard to the choice of pattern documentation format (e.g. Gamma, Alexander, Portland), we are establishing and introducing terminology rules uniting the characteristics of the format we adopted and those belonging to other formats as part of the DTD (Data Type Definition) we defined.

Table 1. A Simplified DTD used by UPADE editor	
XML Notation	
<!ELEMENT pattern (PatternName, AliasNames, ContextUse, UsabilityProblems, Force, Examples, Rationale, DesignSolution, Implementations, Consequence, RelatedPatterns, References)	
<!ENTITY % fieldtext "#PCDATA">	
<!ELEMENT PatternName	(%fieldtext;)*>
<!ELEMENT AliasNames	(%fieldtext;)*>
<!ELEMENT ContextUse	(%fieldtext;)*>
<!ELEMENT UsabilityProblems	(%fieldtext;)*>
<!ELEMENT Forces	(%fieldtext;)*>
<!ELEMENT Examples	(%fieldtext;)*>
<!ELEMENT Rationale	(%fieldtext;)*>
<!ELEMENT DesignSolution	(%fieldtext;)*>
<!ELEMENT Implementations	(%fieldImp;)*>
<!ELEMENT Consequences	(%fieldtext;)*>
<!ELEMENT RelatedPatterns	(%fieldtext;)*>
<!ELEMENT References	(%fieldtext;)*>

FIGURE 4.6: A SIMPLIFIED DTD USED BY UPADE EDITOR

Our idea was sparked by the <peers> concept introduced by the UIML (User Interface Markup Language) language. UIML proposes describing the components of a user interface using general vocabulary, then defining the rules that establish the correspondence between the internal terminology used by UPADE editor and the one used by a given pattern language. By adopting a solution of this sort, pattern writers and users have the option of working in the documentation format of their choice,

while eliminating any terminological ambiguity. Our work at the current time is focused on the definition of terminology rules that enable us also to import and integrate pattern languages such as “Common Ground”, “Experience”, “Brighton”, and “Amsterdam” in the UPADE, in addition to the adopted notation.

4.3.2 XML as a Device-Independent Language for

Implementing Patterns

Given the wide variety of IBIS applications, pattern implementations should exist in various formats. For example, the Web convenient toolbar pattern that provides direct access to frequently used pages such as What’s New, Search, Contact Us, Home Page, and Site Map, can be implemented differently for a Web browser and a Personal Digital Assistant (PDA). For a PDA, this pattern can be implemented as a combo box using the Wireless Markup Language (WML). For a Web browser, it is implemented as a toolbar using embedded scripts or a Java applet in HTML. UPADE should provide advices to pattern users in terms of selecting the suitable implementations for their context. Selection rules should be embedded in the tool.

Furthermore, rather than using different programming languages for coding the different implementations, we are investigating XML as a unified and device-independent language for implementing patterns. By using XML-compliant implementations, patterns can be translated into scripts for script-based environments like HTML authoring tools, beans for Java GUI builders like VisualAge, and pluggable objects like Java applets and ActiveX components. Generating a specific implementation from an XML-based description is now possible because of the

availability of XML-based scripting languages. Among them, the user interface Markup Language, UIML is a potential candidate. UIML descriptions of a user interface can be rendered in HTML, Java, and WML. Tools like the IBM-Automatic code generator from design patterns encourage us to investigate the generation of code from XML-based pattern implementations.

4.4 Summary

In this chapter we described the implementation issue of UPADE application. We demonstrated how UPADE has been implemented by using Java, the JDBC and XML and how these languages made UPADE a usable application for user interface developers.

Chapter 5

Conclusion and Future Work

In the last few years, much activity has been concentrated on discovering and documenting HCI patterns. However, little importance has been placed on how to apply these patterns and combine them to create a complex design. Automating the process of developing patterns-oriented designs can facilitate the usage of patterns while improving the usability of applications. This thesis addressed these challenges. He mainly showed how user interface developers could reuse the existing well- documented HCI patterns to create a complex design. The proposed UPADE prototype allows the developer to define HCI patterns, and compose them. The developer also can import a previously defined HCI pattern, modify it and reuse it again and again to define new ones. Therefore, we can say that UPADE allows user interface developers to reuse part or complete design over and over.

However, as discussed in this thesis, pattern engineering automation tools are useful tools as long as designers who are highly skilled in HCI patterns use them wisely. For this reason, we tried in UPADE to abate disparity between HCI pattern descriptions and their implementation. It is the developer responsibility to assess if the

implementations suggested by UPADE will be applied correctly or not. If user interface developer needs are more specialized, then of course they need to develop the code.

Compared to existing user interface development tools such as GUI builders, UPADE has benefits and detriments. Some of the UPADE are:

- *Facilitates the process of browsing and searching HCI patterns.* UPADE provides user interface developers and HCI patterns developer with just-in-time details and information on both patterns and the process of combining them. The “Search” feature aims to accelerate finding about specific HCI pattern and will reduce the time needed to design a user interface application.
- *Facilitates the process of gluing patterns.* UPADE helps developers to glue HCI patterns and reuse well-developed and documented HCI patterns to design a complex system.
- *Supports pattern modification and creation.* UPADE helps developers to create their own patterns or modify an existing pattern. All the modified patterns can be saved for reuse in other designs.
- *Code generation.* UPADE aims to support the generation of programs from a high level description of patterns

The UPADE prototype we developed has been tested on several sample programs to establish a base-level confidence that it operates correctly. Naturally, extensive testing would be required.

An other interesting issue in this context is the number of well-known HCI patterns. User interface developer will have some limitation to compose the pattern to

An other interesting issue in this context is the number of well-known HCI patterns. User interface developer will have some limitation to compose the pattern to develop a complicated user interface application. Therefore, there is a potential to search new HCI patterns and add them to the UPADE database for later usage.

Another important issue is the code generation. We need more time to finish our own Markup Language. We can cope with these limitations where feasible by expanding and refining our reusable knowledge base as we gain more experience. Nonetheless, the analysis for heuristic guidelines is rather human-dependent, and the certainty of compliance is much less. And finally, Architectural models like patterns are abstract; however, they have many possible ways for implementation. We try to captures many of these possibilities, but conformance checks cannot be absolutely guaranteed.

References

- Alexander C., Ishikawa S., Silverstein M.** (1977). A Pattern Language: Towns, Buildings, Construction. Oxford University Press; (1977)
- Borchers J. O.** (2001).A Pattern Approach to Interaction Design. John Wiley & Sons; first edition (May 16, 2001).
- Budinsky, F., M. Finnie, J. Vlissides, and P. Yu** (1996). Automatic Code Generation from design Patterns [Internet journal] January 4, 1996. IBM Systems Journal, Vol 35, No. 2. Available from <<http://researchweb.watson.ibm.com/journal/sj/352/budinsky.html>> [Accessed June 9th, 2002].
- Casaday, G.** (1997) Notes on a pattern language for interactive usability [Internet Publications] ACM,1997. Available from <http://www.acm.org/sigchi/chi97/proceedings/short-talk/gca.htm> [Accessed January 11th, 2003].
- Cay S. Horstmann, Gary Cornell** (1999). Core Java 2 , Volume 2: Advanced Features (4th Edition). New Jersey ,Prentice Hall PTR; Book and CD-ROM edition (December 27, 1999).
- Gamma E., Helm R., Johnson R. and Vlissides J.** (1994) Design Pattern: Elements of Reusable Object-Oriented Software. Addison-Wesley, Reading, MA.
- Keerthi K Arutla,** (2000).Tool Support For Pattern Oriented Analysis And Design. Master thesis, Department of Computer Science and Electrical Engineering. Morgantown, West Virginia 2000.

MagicDraw (2003). Introducing MagicDraw. Product info, 7-April-2003. Available from <<http://www.magicdraw.com/>> [Accessed April 15th, 2003].

Mahemoff, M. J. and Johnston, L. J. (1998), Pattern Languages for Usability: An Investigation of Alternative Approaches, Asia-Pacific Conference on Human Computer Interaction (APCHI) 98.

Meijers M, (1996) Tool Support for Object-Oriented Design Patterns, Master's Thesis, Utrecht University, CS Dept, INF-SCR-96-28, August 1996. Available from <<http://www.serc.nl/people/florijn/work/patterns.html> > [Accessed August 2th, 2002].

Meijler, T. D., S. Demeyer, and R. Engel (1997). Making Design Patterns Explicit in FACE, A Framework Adaptive Composition Environment, in Software Engineering Notes, ESEC/FSE, Vol. 22, No 6, Nov 1997, pp94-110.

ModelMaker (2002), The Borland Delphi Productivity / CASE tool Borland technology, 1994-2003. Available from <http://www.modelmakertools.com/mm_what_others_say.htm> [Accessed April 22nd, 2003].

Nielsen J. Norman N. (2001) User Interface Architecture. IBM; Second Edition (December, 2001).

OTW, (2001). Object Technology Workbench supports C++, Java, Delphi, as well as SQL-DDL and CORBA-IDL, OTW Software, Inc. Available from <<http://www.otwsoftware.com/english/otw/product.shtml> > [Accessed August 20th, 2002].

Pagel, B., and M. Winter (1996). Towards Pattern-Based Tools, EuroPloP preliminary Conference 1996 Available from <<http://www.informatik.fernuni-hagen.de/import/pi3/publikationen/abstracts/EuroPloP96.pdf>> [Accessed August 12th, 2002].

PCML (2001). Pattern and Component Markup Language (PCML) specifications. ObjectVenture Inc. Draft 3,(2001-2002) Available from <<http://www.objectventure.net/files/docs/PCMLSpecification.pdf> > [Accessed Nov 21st, 2002].

Schütze M., Riegel J. P., Zimmermann G. (1999). PSiGene - A Pattern-Based Component Generator for Building Simulation. Journal Theory and Practice of Object Systems (TAPOS), Vol. 5, No. 2, pp. 83-95, 4 1999

Sefika ,M., Sane A., Campbell R. (1996).Monitoring Compliance of a Software System with its high-Level Design Models, Proc. Of ICSE'96,1996. Available from <<http://choices.cs.uiuc.edu/sane/patlint.pdf>> [Accessed August 2th, 2002].

SOFTEAM (2001). Objectteering C++ Developer. Objectteering Software inc. Available from < <http://www.objectteering.com/products.php>> [Accessed April 20th, 2003].

Suzuki J. and Yamamoto Y. (1998). UML Exchange Format & Pattern Markup Language. UML'98 (June 1998) Available from <<http://www.yy.ics.keio.ac.jp/~suzuki/project/uxf/>> [Accessed Nov 20th, 2002].

Tidwell, J. (1999). COMMON GROUND: A Pattern Language for Human-Computer Interface Design. [Internet journal] 17th May 1999. Available from <http://www.mit.edu/~jtidwell/interaction_patterns.html> [Accessed January 9th, 2003].

The Brighton Usability Group (2001). The Brighton Usability Pattern Collection. [Internet].the University of Brighton, UK Available from <<http://www.cmis.brighton.ac.uk/research/patterns/home.html> > [Accessed Feb 2th, 2002].

Traetteberg H. and Welie M. (2000). Interaction Patterns in User Interfaces (July 2000)
Available from < <http://www.cs.vu.nl/~martijn/patterns/PLoP2k-Welie.pdf> > [Accessed
April 23th, 2003].

Yacoub, S. and H. Ammar (1999). Tool Support for Developing Pattern-Oriented
Architectures, Proceedings of the 1st Symposium on Reusable Architectures and
Components for Developing Distributed Information Systems, RACDIS'99, Orlando,
Florida, August 2-3, 1999

Yacoub S., Xue H., and Ammar, H. (2000). Automating the Development of Pattern-
Oriented Designs. Department of Computer Science and Electrical Engineering, West
Virginia University, Application-Specific Systems and Software Engineering
Technology, ASSET 2000, Richardson, Texas, March 24-25, 2000