ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THESIS PRESENTED TO
ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF DOCTOR OF PHILOSOPHY
Ph.D.

BY
TALEB, Mohamed

A PATTERN-ORIENTED AND MODEL-DRIVEN ARCHITECTURE FOR INTERACTIVE SYSTEMS

MONTREAL, DECEMBER 11, 2008

© Mohamed Taleb, 2008

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Alain Abran, Thesis Director
Software Engineering & Information Technology Department
École de technologie supérieure

Mr. Ahmed Seffah, Thesis Co-director
Computer Science Department, Concordia University

Me. Véronique François, President of the Board of Examiners
Electrical Engineering Department, École de technologie supérieure

Mr. Hakim Lounis
Computer Science Department, Université du Québec À Montréal

Mr. Éric Lefebvre
Software Engineering & Information Technology Department
École de technologie supérieure

THIS THESIS WAS PRESENTED AND DEFENDED

BEFORE A BOARD OF EXAMINERS AND PUBLIC

DECEMBER 15, 2008

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

I want to thank, first of all, my research co-supervisors, Dr. Alain Abran and Ahmed Seffah.

Thanks to **Dr. Abran** for accepting me as a Ph.D. student and, more importantly, trusted me to complete this thesis. I remember very well our first meeting. Immediately, it took on a very friendly and harmonious tone. Alain Abran is a very accessible and cordial person. I thank him a thousand times for his invaluable advice, his collaboration and his rapid, efficient and expert intervention that allowed me to improve my research skills.

To Dr. **Seffah**, I thank him so much for his help, for his guidance and for his contribution my work on his research team on software engineering at Concordia University. Our relation took a harmonious, friendly and solid path quickly and it has been such a pleasure to evolve in a climate of confidence and motivation where what can be often considered as drudgery, was transformed into a pleasant and satisfying experience within a good research environment. Thanks to him, I could learn and understand what research is. I admit that it was not easy and it was really necessary to have a great deal of patience.

I also extend special thanks to Olivier Alnet who participated in the development of the POMA architecture example.

Very special thanks as well to my mother, my father, my brother and his family and to all other members of my family who have endured and supported me during all of these years of study and especially to have encouraged me morally and financially. I owe them all my gratitude and I thank you all again. I dedicate to them this thesis as a proof of my appreciation and love.

UNE ARCHITECTURE ORIENTÉE PATTERN ET DIRIGÉE PAR DES MODÈLES POUR LES SYSTÈMES INTERACTIFS

TALEB, Mohamed

RÉSUMÉ

La pratique quotidienne montre qu'il ne suffit pas d’aborder une conception équipé de directives et de guides conceptuels. Les développeurs logiciels se doivent d’être en mesure d'appliquer des solutions éprouvées extraites des meilleures pratiques de conception. Sans cela, le concepteur ne peut appliquer correctement les directives ni tirer pleinement profit de la puissance de la technologie et cela sans compromettre la qualité du produit final : performance, évolutivité, utilisabilité ou facilité d'utilisation. De plus, le concepteur ne peut « réinventer la roue » à chaque fois qu’il implémente une solution de conception dans un projet ou contexte particulier.

La réutilisation de solutions éprouvées permet de résoudre un certain nombre de problèmes de conception tels que : (1) le découplage des divers aspects des systèmes interactifs (par exemple, l'architecture de la logique de sujet, de l'interface utilisateur, de la navigation et de l'information; et (2) la séparation des aspects spécifiques liés aux plates-formes des caractéristiques communes à tous les systèmes interactifs.

Cette thèse identifie une liste de patterns et différents modèles visant à fournir, sous forme d’une architecture orientée patterns et dirigée par des modèles, une solution globale et intégrative. Les modèles de patterns couvrent plusieurs niveaux d'abstraction, tels que: domaine, tâche, dialogue, présentation, et layout. L’architecture montre comment plusieurs modèles peuvent être combinés à différents niveaux d'abstraction dans des structures hétérogènes, qui peuvent alors être utilisés comme éléments de base dans le développement des systèmes interactifs.

Ce document est divisé en six chapitres. Le premier chapitre présente l’état de l’art sur les « Patterns » en général et sur les différentes architectures de développement pour les systèmes interactifs telles que «les architectures N-tiers», «Pattern-Oriented Design (POD)», «Pattern-Supported Approach (PSA)», et «Model-Driven Architecture (MDA)». Le deuxième chapitre introduit la problématique de la recherche avec ses objectifs, ses limites, la méthodologie de la recherche et ses étapes de la recherche. Le troisième chapitre décrit principalement les parties les plus importantes de la recherche qui est de développer une nouvelle architecture appelée architecture orientée pattern et dirigée par des modèles (***POMA***) pour faciliter le développement des systèmes interactifs incluant ses fondements et ses concepts clés, sa vue d'ensemble, ses justifications versus les architectures N-tiers, POD, PSA, MDA, et ses spécifications. Le quatrième chapitre décrit les niveaux architecturaux et les catégories de patterns utilisés dans POMA. Le cinquième chapitre décrit les catégories de modèles utilisés dans POMA. Le sixième chapitre présente une étude de cas exploratoire appliquée à l'architecture proposée dans cette recherche. Le dernier chapitre présente une conclusion sur les travaux de recherche et son évolution dans le futur.

Mots-clés: Pattern-Oriented and Model-driven Architecture (POMA), Pattern, Model, Interactive System, Software Engineering, Usability, MDA, POD, PSA, N-tiers.

PATTERN-ORIENTED AND MODEL-DRIVEN ARCHITECTURE FOR INTERACTIVE SYSTEMS

TALEB, Mohamed

ABSTRACT

Day-to-day experiences suggest that it is not enough to approach a complex design equipped with design tips, guidelines, and hints. Developers must also be able to use proven solutions emerging from the best design practices to solve new design challenges. Without these, the designer is unable to properly apply guidelines or take full advantage of the power of technology, resulting therefore in poor performance, poor scalability, and poor usability. Furthermore, the designer might "reinvent the wheel" when attempting to implement a design solution.

A number of design problems continue to arise, such as: (1) decoupling the various aspects of interactive systems (for example, business logic, the UI, navigation, and information architecture) and (2) isolating platform specifics from the concerns common to all interactive systems.

In the context of a proposal for a **P**attern-**O**riented and **M**odel-driven **A**rchitecture (POMA) for interactive systems, this thesis identifies an extensive list of pattern categories and types of models aimed at providing a pool of proven solutions to these problems. The models of patterns span several levels of abstraction, such as domain, task, dialog, presentation and layout. The proposed POMA architecture illustrates how several individual models can be combined at different levels of abstraction into heterogeneous structures which can then be used as building blocks in the development of interactive systems.

This document is divided into six chapters: the first chapter presents a background and related work on "Patterns" in general and on various architectures for interactive systems development such as "N-tiers architectures", "Pattern-Oriented Design" (POD), "Pattern-Supported Approach" (PSA), and "Model-Driven Architecture" (MDA). The second chapter introduces the research topic with its objectives, its limits, the research methodology, and research steps. The third chapter describes primarily the most important parts of the research which is the development of a new architecture called **P**attern-**O**riented and **M**odel-Driven **A**rchitecture, facilitating the development of interactive systems including fundamentals and key concepts, an overview, justifications versus N-tiers, POD, PSA, and MDA architectures and specifications. The fourth chapter describes architectural levels and categories of patterns used in POMA. The fifth chapter describes the categories of models used in POMA. The sixth chapter presents an exploratory case study applied to the architecture proposed in this research. The last chapter presents a conclusion on this research work and its expected evolution in the future.

Key words: Pattern-Oriented and Model-driven Architecture (POMA), Pattern, Model, Interactive System, Software Engineering, Usability, MDA, POD, PSA, N-tier.

TABLE OF CONTENTS

Page

INTRODUCTION. 1

CHAPTER 1 BACKGROUND AND RELATED WORK.......... 12
1.1 Patterns.......... 13
    1.1.1 Definition 16
1.2 Models.......... 20
1.3 Architectures.......... 21
    1.3.1 N-tiers.......... 21
    1.3.2 Pattern-Oriented Design (POD) 29
        1.3.2.1 Overview.......... 29
        1.3.2.2 Composition techniques.......... 30
        1.3.2.3 Other techniques.......... 33
    1.3.3 Pattern Supported Approach (PSA) 34
    1.3.4 Model-Driven Architecture (MDA) 36
1.4 Why Combine Patterns and Models?.......... 43
1.5 Summary of chapter.......... 45

CHAPTER 2 RESEARCH ISSUES.......... 47
2.1 Research Goal and Objectives.......... 47
2.2 Research Scope.......... 48
2.3 Research Methodology.......... 49
2.4 Summary of chapter.......... 51

CHAPTER 3 POMA: PATTERN-ORIENTED AND MODEL-DRIVEN ARCHITECTURE 50
3.1 Key concepts of POMA.......... 50
3.2 POMA Overview.......... 56
3.3 POMA justifications.......... 58
3.4 POMA specifications and representation.......... 58
    3.4.1 The eXtensible Markup Language (XML) notation.......... 58
    3.4.2 The Unified Modeling Language (UML) notation 60

CHAPTER 4 PATTERNS IN POMA 63
4.1 Patterns and Pattern-Oriented Architecture 63
4.1.1 Architectural Levels and categories of patterns 63
4.1.1.1 Information patterns 65
4.1.1.2 Interoperability patterns 66
4.1.1.3 Visualization patterns 67
4.1.1.4 Navigation patterns 68
4.1.1.5 Interaction patterns 69
4.1.1.6 Presentation patterns 71
4.1.2 Patterns Composition 72
4.1.3 Pattern mapping 79
4.2 Summary of chapter 84

CHAPTER 5 MODELS IN POMA 85
5.1 Model Categorizations 85
5.1.1 Domain model 86
5.1.2 Task model 87
5.1.3 Dialog model 88
5.1.4 Presentation model 89
5.1.5 Layout model 90
5.2 Model Transformation 91
5.3 Source code generation 92
5.4 Summary of chapter 92

CHAPTER 6 CASE STUDY 91
6.1 Overview 91
6.2 Defining the Domain Model 96
6.3 Defining the Task Model 103
6.4 Defining the Dialog Model 113
6.5 Defining the Presentation and Layout Models 119

CONCLUSION 131

APPENDIX I GLOSSARY OF TERMS 140

APPENDIX II TECHNICAL REPORT OF INTERACTIVE SYSTEM DEVELOPMENT TOOLS: TRENDS AND CHALLENGES IN INTERACTIVE SYSTEM DEVELOPMENT TOOLS: REQUIREMENTS FOR PATTERN-ORIENTED AND MODEL-BASED ARCHITECTURE 14645
1. Introduction 14645
2. Content Management System (CMS) 149
2.1 Example of tools for CMS 151
    2.1.1 Zope: Tools available for accessing Zope 151
    2.1.2 PhPNuk 153
2.2 List of Tools in a Content Management Systems 154
3. Tools for Model-Based Approach and for Patterns 158
3.1 Definitions and Advantages of Model-Based UI Development 158
    3.1.1 Definition of Model-Based UI Development 158
    3.1.2 Advantages of Model-Based UI Development 159
    3.1.3 Different models 159
3.2 Different tools for Model-Based Approach 160
3.3 Pattern-Oriented Tools 163
4. Formalisms and notations for patterns, architectures and models specifications 164
4.1 Different formalisms and languages 164
4.2 Examples of different languages and notations 165
    4.2.1 User Interface Markup Language (UIML) 165
    4.2.2 eXtensible User Interface Language (XUL) 166
    4.2.3 eXtensible Interface Markup Language (XIML) 166
    4.2.4 Existing Model-Based Framework 168
5. References 168

APPENDIX III EXAMPLE OF XML SOURCE CODE FOR POMAML STRUCTURAL NOTATION 174

LIST OF APPENDICES 178

BIBLIOGRAPHY 179

LIST OF TABLES

Page

Table 1.1 Description of architectural levels of patterns……………………………...24

Table 1.2 Pattern-Oriented generic classification schema for POMA Architecture…..28

Table 1.3 Summary of architectures (N-tiers, POD, PSA, MDA) and their characteristics assessment……………………………………………………….46

Table 4.1 Architectural levels, categories of patterns and examples…………………..63

Table 6.1 Pattern Summary…………………………………………………………....95

Table 6.2 Example of pattern mapping of the Domain model for laptop and PDA platforms……………………………………………………………………97

Table 6.3 Example of pattern mapping of Task model for laptop and PDA platforms…………………….……………………………………………108

Table 6.4 Example of pattern mapping of Dialog model for laptop and PDA platforms…………………………………………………………………...115

Table 6.5 Example of pattern mapping of the Presentation model for laptop and PDA platforms…………………………………………………………………...122

Table 6.6 Example of pattern mapping of the Layout model for laptop and PDA platforms…………………………………………………………………...128

LIST OF FIGURES

Page

Figure 1.1 Example of an Alexander pattern.. 15

Figure 1.2 Main elements of a pattern..17

Figure 1.3 Class Diagram of Model-View-Controller architectural pattern............. 22

Figure 1.4 Core J2EE 5-tier architectural level of patterns...............................23

Figure 1.5 UML Class Diagram of Core J2EE Patterns Architecture......................25

Figure 1.6 Patterns-Oriented Design architecture...31

Figure 1.7 The PSA architecture with the relationships between PSA patterns......... 35

Figure 1.8 Transformations of MDA architecture..38

Figure 1.9 PIM, PSM and Implementation... 40

Figure 1.10 Foundational Concepts of the MDA..41

Figure 2.1 Methodology Research.. 50

Figure 3.1 Key concepts of POMA..54

Page

Figure 3.2 POMA architecture for interactive systems development..................... 57

Figure 4.1 Examples of Information Patterns... 66

Figure 4.2 Adapter pattern...67

Figure 4.3 The Navigation Spaces Map pattern implemented using Tree Hyperbolic, a sophisticated visualization technique...68

Figure 4.4 Breadcrumb Pattern.. 69

Figure 4.5 Stepping pattern... 70

Figure 4.6 An example of a grid...71

Figure 4.7 Example of structural patterns: Executive Summary Pattern................ 72

Figure 4.8 Similar Pattern.. 73

Figure 4.9 Two Competitor Pattern...74

Figure 4.10 A Home Page Design Pattern using others patterns.......................... 74

Figure 4.11 UML Class Diagram of Architectural Level and Categories of Patterns for Interactive System.. 76

Figure 4.12 Class structure of POMA's Models and Patterns............................. 77

Figure 4.13 Pattern structure of the POMAML Markup Language....................... 78

Page

Figure 4.14 The Web Convenient Toolbar pattern implementations and Look and Feels for different platforms……………………………………………………..80

Figure 4.15 Examples of patterns…………………………………………………82

Figure 4.16 Migration of the CBC site to a PDA Platform using Pattern Mapping…... 83

Figure 4.17 Pattern-Oriented Composition and Mapping Design Architecture………..83

Figure 6.1 Graphical representation of the pattern……………………………………96

Figure 6.2 UML class diagram of the PIM Domain model…………………………...96

Figure 6.3 UML class diagram of the PSM Domain model for a laptop platform…... 98

Figure 6.4 UML class diagram of the PSM Domain model for PDA platform……… 99

Figure 6.5 The *Login* pattern on the laptop platform………………………………… 100

Figure 6.6 The *Login* pattern on the PDA platform……………………………........... 100

Figure 6.7 Login view of the interactive system on the laptop platform…………….. 101

Figure 6.8 Login view of the interactive system on the PDA platform……………… 101

Figure 6.9 Task model of the environmental management interactive system…………………………………………………………………….. 105

Figure 6.10 UML class diagram of the PIM Task model………………………………107

Page

Figure 6.11 UML class diagram of the PSM Task model mapped for a laptop platform……………………………………………………………………109

Figure 6.12 UML class diagram of the PSM Task model mapped for a PDA platform 110

Figure 6.13 UML class diagram of a PIM Dialog Model……………………………... 114

Figure 6.14 Graph structure suggested by the Wizard pattern………………………… 114

Figure 6.15 UML class diagram of the PSM Dialog model for a laptop platform……. 116

Figure 6.16 UML class diagram of the PSM Dialog model for a PDA platform……... 117

Figure 6.17 Dialog Graph of the environmental management interactive system for laptop and PDA platforms…………………………………………………….. 118

Figure 6.18 UML class diagram of a PIM Presentation model………………………...121

Figure 6.19 UML class diagram of the PSM Presentation model for a laptop platform.123

Figure 6.20 UML class diagram of the PSM Presentation model for a laptop platform.124

Figure 6.21 UML class diagram of a PIM Layout model……………………………... 127

Figure 6.22 UML diagram of PSM Layout Model for a Laptop platform……………... 129

Figure 6.23 UML diagram of PSM Layout Model for a PDA platform………………… 130

Figure 6.24 Screenshot of the Environmental Management Interactive System for a Laptop platform………………………………………………………………... 131

LIST OF ABREVIATIONS AND ACRONYMS

CIM	Computation Independent Model
CIV	Computation Independent Viewpoint
GUI	Graphical User Interface
HCI	Human Computer Interaction
HTML	Hypertext Markup Language
IEEE	Institute of Electrical and Electronics Engineers, Inc
ISML	Interface Specification Meta-Language
JDBC	Java Databases Connectivity
JDK	Java Development Kit
JFC	Java Foundation Classes
MDA	Model-Driven Architecture
MPML	Model Pattern Markup Language
MVC	Model – View - Controller
PIM	Platform Independent Model
PIV	Platform Independent Viewpoint
POA	Pattern-Oriented Architecture

POD	Pattern-Oriented Design
POMA	Pattern-Oriented and Model-driven Architecture
POMAML	Pattern-Oriented and Model-driven Architecture Markup Language
PSA	Pattern-Supported Approach
PSM	Platform Specific Model
PSV	Platform Specific Viewpoint
SEI	Software Engineering Institute
TCP/IP	Transfer Control Protocol / Internet Protocol
UI	User Interface
UCD	User Centered Design
UML	Unified Modeling Language
XML	eXtensible Markup Language
WAP	Wireless Access Protocol
WSDL	Web Service Definition Language

INTRODUCTION

A. Interactive systems

In Software Engineering, an "Interactive System" is a system accessed by interfaces over a network such as the Internet, intranet, extranet or by a traditional medium.

Interactive systems are popular due to the ubiquity of the browser for the client, sometimes called a thin client. The ability to update and maintain interactive systems without distributing and installing systems on potentially thousands of client computers is a key reason for their popularity. Interactive systems are used to implement, for example, Webmail, online retail sales, online auctions, wikis, discussion boards, Weblogs, MMORPGs and a number of other functions. The scope of this research project is therefore limited to interactive systems.

In short, an interactive system is a program with which the user engages in conversation (dialog) in order to accomplish tasks. An interactive system consists of two parts: the software part, which is referred to as the interactive application; and the hardware part which supports the execution of the software. The software can, in turn, be divided into two sub-parts: the user interface, and the algorithmic, which is the semantics of the interactive application. The hardware in an interactive system consists of input and output devices and various managers (drivers of devices) that provide the physical support to the execution of the interactive application.

At the same time, a user interface (UI) can be seen as a means by which the user and the machine can exchange data. For example, the screen on which data are displayed is a medium for user-machine interaction and for feedback in response to the user's actions. Therefore, a UI is part of an interactive application which:

- Presents the output to the user;
- Collects the user's inputs and transmits them to interactive systems which treat them;
- Handles the sequence of dialogs.

Over the past two decades, research on interactive systems and user interfaces (UI) engineering has resulted in several architectural models which constitute a major contribution not only to facilitate the development and maintenance of interactive systems, but also to promote the standardization, portability and ergonomic "usability" (ease of use) of the interactive systems being developed. Such architectures provide:

- A precise definition of the UI aimed at: (i) presenting the output to the user; (ii) gathering user entries to transmit them to the interactive system procedures that will treat them; (iii) handling the dialog sequence;
- The separation of concerns, especially the decoupling of the UI from the system semantics;
- The definition of reusable and standardized UI components;
- The decentralization of the dialog management, help, and errors across the various components of an interactive system;
- Programming driven by events.

B. Architectures overview

Buschmann *et al.* (1996) define architectural models as: "*the structure of the subsystems and components of a system and the relationships between them typically represented in different views to show the relevant functional and non functional properties.*" This definition introduces the main architectural components (for instance, subsystems, components and connectors) and covers methods to represent them, including both functional and non-functional requirements, by means of a set of views.

Bass et al. (2003) define the software architecture as: "*The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them*". The architecture defines the components (such as modules, objects, processes, subsystems, compilation units) and the relevant relations (such as "calls", "sends data to", "synchronizes with", "uses", "depends on", "instantiates"). The architecture is the result of early design decisions that are necessary before a group of people can collaboratively build a software system (Bass et al., 2003).

A number of architectures specific to interactive systems have been proposed, e.g., Seeheim model (Pfaff, 1985) and (Green, 1985), Model-View-Controller (MVC) (Goldberg, 1984), Agent Multi-Faceted (AMF) (Ouadou, 1994) which is an extension of MVC, Arch/Slinky (Gram and Cockton, 1996), Presentation Abstraction Control (PAC) Coutaz, 1987) and (Coutaz, 1990), PAC-Amadeus and Model-View-Presenter (MVP) (Bass et al., 2003). Most of these architectures consist of three main elements: (1) abstraction or model, (2) control or dialog and (3) presentation. Their goal is to improve and facilitate the design of interactive systems. However, even though the principle of separating an interactive system into components has its design merits, it can also be a source of serious adaptability and usability problems in systems which provide fast, frequent, and intensive semantic feedback: the communication between the view and the model makes the interactive system highly coupled and complex.

Among the weaknesses of these architectures, one can mention:

- No guidance is provided to encourage the designer to cope with the different aspects of the dialog such as assistance or error-handling;
- Lack of provisions to deal with the constraints for the design and description of the interface, when these constraints are of great importance to the designer (Myers, 1989a), (Myers, 1989b), (Myers, 1989c), (Myers, 1990), (Myers et al., 1990) and (Darses, 1990);
- The architectural models are poorly located in relation to the life cycle of the UI, which can lead, in particular, to difficulties concerning the passage of the problem analysis (analysis of user needs), expressed generally in terms of tasks and interaction sequences, and to the concepts put forward by these architectures (agents, presentation components, dialog components).

C. About Patterns

Patterns have been proposed to alleviate some of these weaknesses, and indeed were introduced based on the observation given by Alexander et al. (1979) in section 1.1.1. Such a pattern provides, on a single level, a pool of proven solutions to many of the recurring weaknesses listed above.

The Pattern-Oriented Software Architecture (Schmidt et al., 2000) is an example of a new approach which combines individual patterns into heterogeneous structures and, as such, can be used to facilitate a constructive instantiation of a system architecture.

Patterns have proven their utility in different fields of application. Design patterns (Gamma et al., 1995) or architectural patterns (Buschmann et al., 1996) are well known uses of successful patterns in computing.

Patterns provide various benefits, such as:

- Well-established solutions to architectural problems;
- Help in documenting architectural design decisions;
- Facilitation of communication between users through a common vocabulary;
- A common interlingua (or lingua franca) (Erickson, 2000);
- Documentation of problems and their corresponding best solutions.

However, one notes that the emergence of patterns in the architectural development of interactive systems has not solved some of the problems associated with this development.

Among the challenging problems addressed in this thesis are the following:

(a) Decoupling of the various aspects of interactive systems such as business logic, user interface, navigation, and information architecture,
(b) Isolation of the platform-specific problems from the concerns common to all interactive systems.

D. About Models

In 2001, the Object Management Group introduced the Model-Driven Architecture (MDA) initiative as an architecture to interactive system specification and interoperability based on the use of formal models (i.e. defined and formalized models). The main idea behind MDA is to specify business logic in the form of abstract models. These models are then mapped (partly automatically) to different platforms according to a set of transformation rules. The models are usually described in UML in a formalized manner, which can be used as input for tools to perform the transformation process.

Indeed, a model is a formal description of some key aspects of an interactive system, from a specific viewpoint. As such, a model always presents an abstraction of the "real" thing, by ignoring or deliberately suppressing those aspects that would not be of interest to a user of that model. In other words, a model is the main element of the system. Different modeling constructs focus attention by ignoring certain things (D'Souza, 2001). For example, an architectural model of a complex interactive system might focus on its concurrency aspects, while a financial model of a business might focus on projected revenues. Model syntax includes graphical or tabular notations and text.

D'Souza (2001) has identified key opportunities and modeling challenges and he has illustrated how "**M**odel" and "**A**rchitecture" could be used to enable large-scale model-driven integration. The advantages of the models are as follows:

- Validation of the correctness of a model is made easier;
- Production implementations on multiple platforms is easier;
- Integration / interoperability across platforms is better defined;
- Generic mappings / patterns can be shared by many designs;
- Models constitute an interactive system of tool-supported solutions.

However, one notes that model-driven architecture has some weaknesses as well:

- MDA does not provide a standard for the specification of mappings: different implementations of mappings can generate very different codes and models which can create dependencies between the interactive system and the mapping solution used;
- Designers must take into account a diversity of platforms which exhibit drastically different capabilities. For example, Personal Digital Assistants (PDAs) use a pen-based input mechanism and have an average screen size in the range of 3 inches;
- The architectural models must be located and compared to the life cycle of the UI, in particular, difficulties may arise related to the problem analysis (analyzing user needs).

> expressed generally in terms of tasks and interaction sequences, and to the concepts proposed by these architectures (agents, presentation components, and dialog components).

Models should be precise enough to at least enable unambiguous communication analysis and abstract enough to focus attention and provide insights. A model is simpler to understand than the thing it represents; well-structured models can make complex interactive systems understandable. Modeling helps users achieve consensus about what exists or can be built, since it provides a focus on which to agree or disagree. A good model does not have to be executable, but it must be readily validated against examples.

Models are commonly used to represent the flexibly of complex interactive systems. Models can be viewed at many levels of abstraction, and complementary model views can be combined to give a more intelligible and accurate view of a system than a single model alone. Meservy and Fensternacher (2005) claim that many software development experts have long advocated using models to understand the problem that a system seeks to address; yet development teams commonly employ models only in the early stages of modeling. Often, once construction begins, the teams leave these models behind and never update them to reflect their design changes during the project.

Most software developers would agree that modeling should play a role in every project (Meservy and Fensternacher, 2005). However, there is no clear consensus on what that role should be, how developers should integrate modeling into other development activities and who should participate in the modeling process (Meservy and Fensternacher, 2005).

E. New generation of platforms in interactive systems

In recent years, interactive systems have matured from offering simple interface functionality to providing intricate processes such as end-to-end financial transactions. Users have been given more sophisticated techniques to interact with available services and information using different types of computers. Different kinds of computers and devices (including, but not limited to, traditional office desktops, laptops, palmtops, PDAs with and without keyboards, mobile telephones, and interactive televisions) are used for interacting with such systems. One of the major characteristics of such cross-platform interactive systems is that they allow a user to interact with the server-side services and contents in various ways. Interactive systems for small and mobile devices are resource constrained and cannot support a full range of interactive system features and interactivity because of the lack of screen space or low bandwidth.

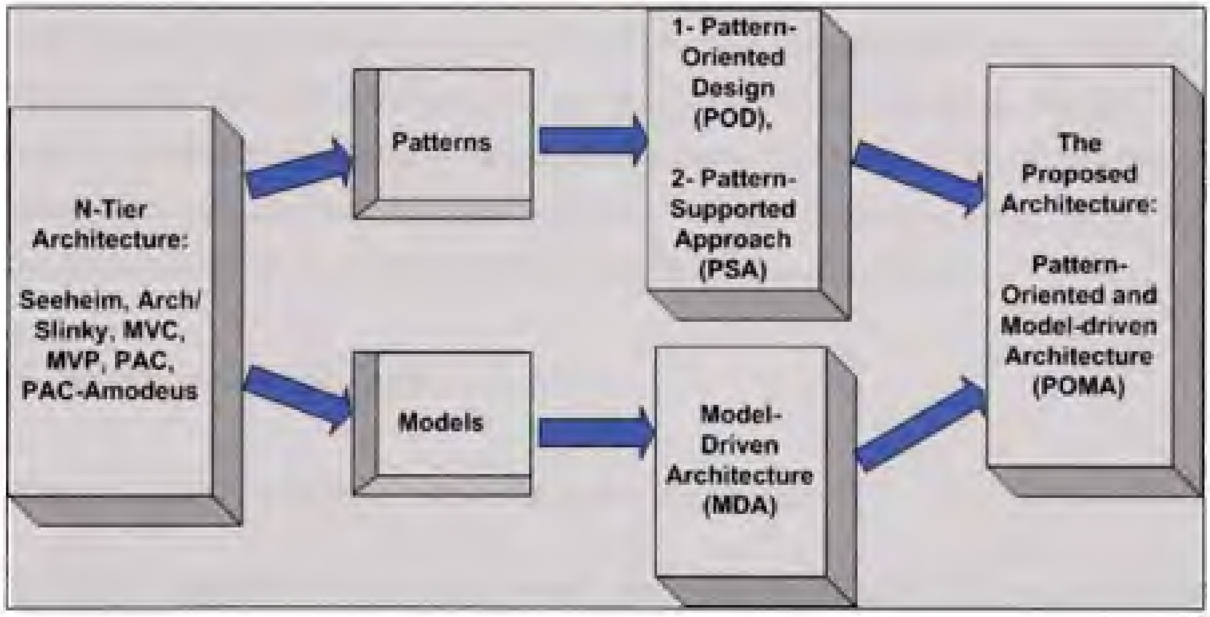
The mosaic of interactive systems and multiple platforms has led to the emergence of interactive systems as a sub-discipline of software engineering with some specific challenges. One important question is how to develop and deploy the same system for different platforms – without "architecturing" and specifically writing code for each platform, for learning different languages and the many interactive systems design guidelines that are available for each platform.

The key motivations for this research project are:

- The need to adapt: (1) patterns and some of development pattern-oriented architectures such as POD, PSA, and (2) Model-Driven Architecture to address some of the challenges of designing and developing cross-platform interactive systems;
- The need to support both novices and experts in interactive systems development.

Our research goal can be stated as follows: "Define a new architecture to facilitate the development and migration of interactive systems while improving their usability and quality."

To pursue this goal, it is necessary to define an architecture, supported by a CASE tool, to glue patterns together. In this thesis, some of the fundamentals of such architectures are identified and an architecture called **Pattern-Oriented** and **Model-driven Architecture (POMA)** is presented (Figure 3.2). Presented also is an evaluation of the feasibility of some phases of this architecture, such as composition and mapping patterns and transformation models to create platform-independent models (PIM) and platform-specific models (PSM). The following figure summarizes the architectural patterns and models that were combined to obtain the POMA architecture.


N-Tier Architecture:
Seeheim, Arch/ Slinky, MVC, MVP, PAC, PAC-Amodeus
Patterns
Models
1- Pattern-Oriented Design (POD),
2- Pattern-Supported Approach (PSA)
Model-Driven Architecture (MDA)
The Proposed Architecture:
Pattern-Oriented and Model-driven Architecture (POMA)

Architectural Patterns and Models.

According to the presented figure above, the proposed POMA architecture must rely primarily on the concepts of N-tier architectures, the Pattern-Oriented architectures such as POD and PSA and the Model-Driven Architecture (MDA).

This thesis proposes an architectural model that combines two key approaches: model-driven and pattern-oriented. Firstly, fundamentals, key concepts, an overview, justifications, and specifications of the proposed architecture, called *POMA* are presented, which constitute a development architecture. Secondly, architectural levels and categories of patterns are described as well as the various relationships between patterns. These relationships are used next to combine and map several categories of patterns to create a pattern-oriented design for an interactive system, and to show how to generate specific implementations suitable for different platforms from the same pattern-oriented design. Thirdly, five categories of models (Domain model, Task model, Dialog model, Presentation model, and Layout model) are proposed which address problems such as: (a) decoupling the various aspects of interactive systems, such as business logic, UI, navigation, and information architecture; and (b) isolating platform-specific problems from the concerns common to all interactive systems. Fourthly, the proposed Pattern-Oriented and Model-driven Architecture (POMA) illustrates how the individual models mentioned above can be combined at different levels of abstraction into heterogeneous structures to be used as building blocks in the development of interactive systems. Fifthly, a case study to illustrate the new architecture and its practical relevance is presented. Finally, a conclusion on the research carried out and comments on future evolution is presented.

F. Organization of this Thesis

The organization of the remainder of this thesis is as follows:

In chapter 1, background and related work is discussed. In particular, one reviews patterns, models, and existing architectures such as N-tiers, Pattern-Oriented Design (POD), Pattern-Supported Approach and Model-Driven Architecture (MDA).

In chapter 2, research issues are introduced that lead to the research statement, the research objectives including the research steps, the research scope and the research methodology.

In chapter 3, the results of this research project are presented, describing primarily the most important parts of the research which are: the development of a new architecture called **P**attern-**O**riented and **M**odel-Driven **A**rchitecture (***POMA***) to facilitate the development of interactive systems including its fundamentals and key concepts; its overview; its justifications versus N-tiers, POD, PSA, and MDA architectures; its specifications.

In chapter 4, a detailed description of patterns included in POMA is presented. This POMA employs architectural levels and categories of patterns, pattern composition rules (i.e.. the relationships between patterns considered in this architecture) and the pattern mapping rules that enable one to obtain the final model of the proposed architecture.

In chapter 5, a detailed description of the models included in POMA is presented. This POMA applies the types of models and the model transformation rules, which are applied for each type of model [POMA.PIM] or [POMA.PSM]. These transformation rules enable one to build relations between the models of each category, i.e. the models [POMA.PIM] and [POMA.PSM] of the proposed architecture.

In chapter 6, a case study of a multi-platform interactive system is presented to illustrate and clarify the core ideas of POMA architecture and its practical relevance.

Finally, this work summarizes the key contributions, the implications for software engineering, the practical implications, the limitations and strengths and future avenues for research.

CHAPTER 1

BACKGROUND AND RELATED WORK

This chapter presents the literature review of existing N-tier architectures, the Pattern-Oriented Design (POD), the Pattern-Supported Approach (PSA) and the Model-Driven Architecture (MDA).

The first section introduces the concepts of patterns, and related terminologies. The second section introduces the concepts of models. The third section defines and investigates the concepts of various architectures such as N-tier architectures (MVC, J2EE, and Zachman), Pattern-Oriented Design (POD) architecture, Patterns Supported Approach (PSA), and the basic foundation of the Model-Driven Architecture (MDA) proposed by the (OMG Group, 2008). The fourth section discusses the combining of patterns and models. The fifth section presents an assessment of these development architectures and identifies the differences in these architectures and related research issues.

1.1 Patterns

Christopher Alexander, in the late 1970's, in his two books, *A Pattern Language* (Alexander et al., 1977) and *A Timeless Way of Building* (Alexander, 1979) discusses the capture and use of design knowledge in the format of patterns, and presents a large collection of pattern examples to help architects and engineers in the design of buildings, towns, and other urban entities.

As an illustration, Alexander proposed an architectural pattern called *Wings of Light* (Alexander et al., 1977), where the problem statement is:

> "Modern buildings are often shaped with no concern for natural light - they depend almost entirely on artificial light. But, buildings which displace natural light as the major source of illumination are not fit places to spend the day." (Alexander et al., 1977).

In addition to other information such as design rationale, examples, and links to related patterns, the solution statement is:

> "Arrange each building so that it breaks down into wings which correspond, approximately, to the most important natural social groups within the building. Make each wing long and as narrow as you can - never more than 25 feet wide." (Alexander et al., 1977).

Introduced by (Alexander et al., 1977), design patterns can be viewed as building blocks that may be composed of several components to create designs. A single pattern describes a problem that appears constantly in the environment with a corresponding solution to this problem expressed in a way that allows designers to reuse this solution for different platforms. For cross-platform interactive systems, patterns are interesting for following three reasons. Refer to (Buschmann, 1996) for a more general discussion on patterns and their benefits:

- They come from experiments on established experiences and were not created artificially;
- They are a means of documenting architectures;
- In the case of cross-platform development, they provide for the possibility for a team to have a common vision and language.

Alexander's idea stemmed from the premise that there was something fundamentally incorrect with the approach taken by twentieth century architectural design methods and practices. He introduced patterns as a three-part rule to help architects and engineers with the design of buildings and towns. His definition of a pattern is as follows: "Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution" (Alexander, 1979). The underlying objective of Alexander's patterns was to tackle architectural-related problems that occurred over and over again in a particular environment, by providing commonly accepted solutions. Figure 1.1 illustrates an example of one of Alexander's patterns adapted from (Erickson, 2000). The numbers in parenthesis are identifiers for the patterns.

Street Café (88)

[Picture omitted]

...Neighbourhoods are defined by Identifiable Neighbourhood (14); their natural points of focus are given by Activity Nodes (30) and Small Public Squares (61). This pattern, and the ones which follow it, give the neighbourhood and its points of focus, their identity.

The street cafe provides a unique setting, special to cities: a place where people can sit lazily, legitimately, be on view, and watch the world go by.

The most humane cities are always full of street cafes. Let us try to understand the experience, which makes these places so attractive. We know that people enjoy mixing in public, in parks, squares, along promenades and avenues, in street cafes. The preconditions seem to be: the setting gives you the right to be there, by custom; there are a few things to do that are part of the scene, almost ritual: reading the newspaper, strolling, nursing a beer, playing capture; and people feel safe enough to relax, nod at each other, perhaps even meet. A good cafe terrace meets these conditions. But it has in addition, special qualities of its own: a person may sit there for...

[Nine paragraphs of rationale omitted]

Therefore:

Encourage local cafes to spring up in each neighbourhood. Make them intimate places, with several rooms, open to a busy path, where people can sit with coffee or a drink and watch the world go by. Build the front of the cafe so that a set of tables stretch out of the cafe, right into the street.

[Diagram omitted]

Build a wide, substantial opening between the terrace and indoors-OPENING TO THE STREET (165); make the terrace double as A PLACE TO WAIT (150) for nearby bus stops and offices; both indoors and on the terrace use a great variety of different kinds of chairs and tables-DIFFERENT CHAIRS (251); and give the terrace some low definition at the street edge if it is in danger of being interrupted by street action-STAIR SEATS (125), SITTING WALL (243), perhaps a CANVAS ROOF (244).

[Text omitted]...

Figure 1.1 Example of an Alexander pattern.
(Extracted from (Alexander, 1979) and (Javahery, 2003))

Alexander's patterns are written in narrative form. Even if they do not have clearly defined attributes per se, they are all structured in a specific way with a description of the problem, solution, and context. The idea of using Alexandrian-type patterns as a design tool has been quite influential in a variety of domains in the last decade, including software engineering. In recent years, the Human-Computer Interaction (HCI) community has adopted the idea of patterns for interactive system design (Javahery, 2003).

Patterns have been introduced as a tool to capture and disseminate proven design knowledge, and to facilitate the design of more usable interactive systems. Patterns capture and communicate the best practices for user interface design with a focus on the user's experience and on the context of use. As a result, patterns are attractive tools for User Centered Design (UCD), with interesting ramifications for designing across a variety of contexts (Coram and Lee, 1998) and (Erickson, 2000).

1.1.1 Definition

Patterns are all around us. Patterns can be found in nature, such as bubbles and waves, in buildings and in their windows. Patterns can also be found in software. The term "pattern" is adopted in software engineering from the work of the architect Christopher Alexander, who explored patterns in architecture. Thus, in an attempt to define a pattern, a starting point would be a definition of a pattern given by (Alexander, 1979):

> "Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice" (Alexander, 1979).

Coplien (1998) gives a similar generic definition of a software pattern, as

> "… The thing and the instructions for making the thing".

Other suggested definitions include:

- "The term pattern defines both one thing and how to achieve it" (Coad, 1992).
- "A pattern describes a problem to be solved, a solution, and the context in which this solution is considered. A pattern appoints a technique and describes its costs and benefits, allowing a team to set a common vocabulary to describe models" (Johnson, 1997).
- "The patterns capture the knowledge that experts apply to solve recurring problems" (Rising, 1996).

- "A pattern is an idea that was used in a practical context and which will probably be used by others" (Fowler, 1997).
- "A solution to a problem in a context" (Lea, 1997).

A pattern can be viewed in a prose format for recording solutions, such as, design information, which has worked well in the past and can be applied again in similar situations in the future (Beck et al., 1996). The need for the introduction of patterns in software describes the model needed for an engineering discipline to mature, and like a mature engineering discipline, to provide handbooks for describing successful solutions to known problems.

Every pattern has three main elements: a context, a problem, and a solution (Figure 1.2). The context describes a set of recurring situations in which the pattern can be applied. The problem refers to a set of forces, i.e., goals and constraints, which occur in the context. Generally, the problem indicates when to apply the pattern. The solution refers to a design model or a design rule that can be applied to resolve these problem forces. The solution describes the elements that constitute a pattern, the relationships among these elements, as well as responsibilities and collaboration.

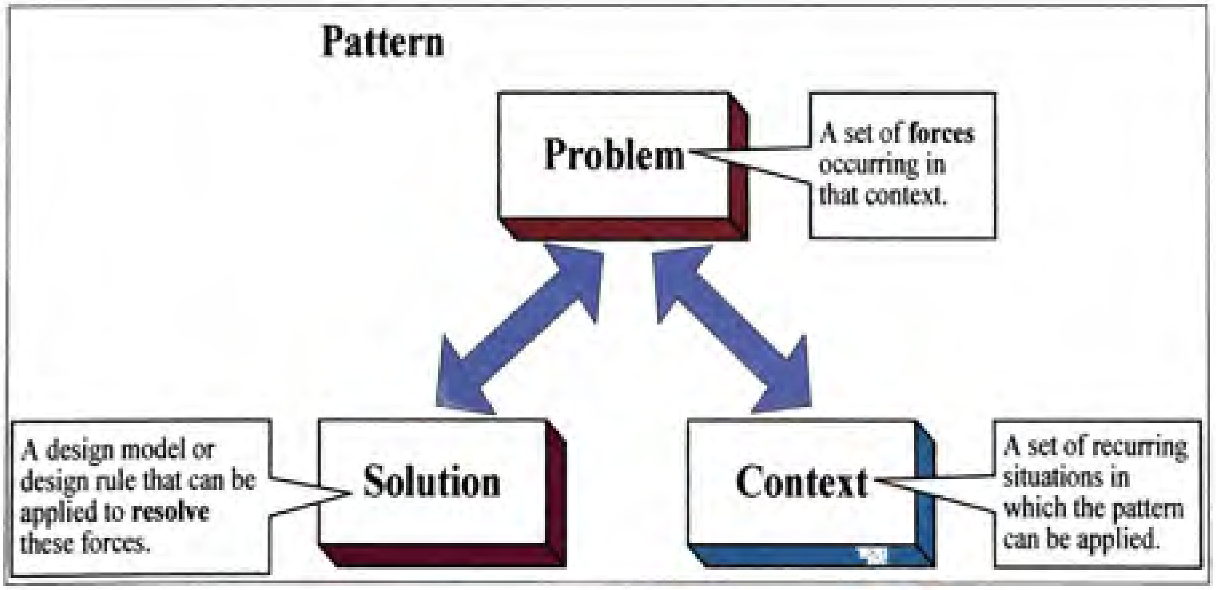

Pattern
Problem
A set of forces occurring in that context.
Solution
A design model or design rule that can be applied to resolve these forces.
Context
A set of recurring situations in which the pattern can be applied.

Figure 1.2 Main elements of a pattern.

The following example from Alexander et al. (1977) illustrates the main elements of a pattern and their relationship.

Window place Consider one simple problem that can appear in the architecture. Let it be assumed that a person wants to be comfortable in a room, implying that the person needs to sit down to really feel comfortable. Additionally, the sunlight may be an issue, since the person is most likely to prefer to sit near the light. Thus, the forces of pattern in this example are:

(i) The desire to sit down;

(ii) The desire to be near the light.

The *solution* to this *problem* might be that in every room, the architect should make a window into a *window place*.

Not every pattern can be considered to be a good pattern. There is a set of criteria that a pattern must meet in order to be good:

- a solution (but not obvious);
- a proven concept ;
- relationships;
- the human component.

A pattern encapsulating these criteria is considered to be a good pattern (Gamma et al., 1995), (Alexander et al., 1977) and (Coplien, 2001). Thus, (Gamma et al., 1995), (Alexander et al., 1977) and (Coplien, 2001) claim, according to the criteria quoted above, that a good pattern should solve a problem, i.e., patterns should capture solutions, not just abstract principles or strategies. A good pattern should be a proven concept, i.e., patterns should capture concrete solutions, not theories or speculation. A good pattern should not provide an obvious solution, i.e., many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns generate a solution to a problem indirectly, which is a necessary approach for the most difficult problems of design. A pattern also describes a relationship, not just modules, but describes deeper system structures and mechanisms. Additionally, a good pattern should contain a significant human component (minimize human intervention). Many softwares serve human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility.

Similar to the entire software engineering community, interactive system engineers and the user interface design community have a forum for discussions on pattern languages for user interface design and usability.

The goals of patterns are:

1. To share successful user interface design solutions among HCI professionals;
2. To provide a common ground for anyone involved in design, development, usability testing;
3. To provide any user a highly interactive system including different types of applications.

A number of pattern languages have been suggested. For example, Van Duyne's (2003) "The Design of Sites", Welie's (1999) Interaction Design Patterns, and Tidwell's (1997) UI Patterns and Techniques play an important role. In addition, specific languages such as Laakso's (2003) User Interface Design Patterns and the UPADE Language (Engelberg and Seffah, 2002) have been proposed as well. Different pattern collections have been published including patterns for Web page layout design (Tidwell, 1997) and (Coram and Lee, 1998) for navigation in large information architectures, as well as for visualizing and presenting information.

The idea of using patterns in interactive system design and engineering is not new. It has its roots in the popular Gang of Four book (Gamma et al., 1995). Different collections of patterns include patterns for user interface design (Tidwell, 1997), (Coram and Lee, 1998) and (Welie, 1999) for navigation in large information architectures as well as for visualizing and presenting information. More recently, the concept of usability patterns has been introduced and discussed as a tool for assuring the usability of the developed systems (CHI, 1999), (INTERACT, 1999) and (UPA, 2001). A usability pattern is a proven solution for a User Centered Design (UCD) problem that recurs in different contexts. The primary goal of usability patterns in general is to create an inventory of solutions to help user interface designers tackle user interface (UI) development problems that are common, difficult and frequently encountered (Loureiro and Plummer, 1999).

1.2 Models

As the complexity of interactive systems grows, the role of models is becoming essential for dealing with the numerous aspects involved in their development and maintenance processes. Models allow the relevant aspects of an interactive system to be captured from a given perspective and at a specific level of abstraction. In a model-driven UI design approach, various models are used to describe the relevant aspects of the UI. Many facets exist, as well as related models. Design is an assembly of parts that realizes a specification. A model of an interactive system is a specification of that system and its environment for certain purposes. Models consist of a set of elements with a graphical and/or textual representation (Koch and Fast, 2006). The idea behind model-driven design is to create different models of an interactive system at different levels of abstraction, and to use transformations of the models to produce the implementation of an interactive system.

Thus, one can define a model as follows:

1. In the MDA, a *model* is a representation of a part of the function, structure and/or behavior of a system.
2. A *model* of an interactive system is a description or specification of that interactive system and its environment for certain purposes. A *model* is often presented as a combination of drawings and text. The text may be in a modeling language or in a natural language.

A number of distinct models have been suggested, for example:

- The OMG's Model-Driven Architecture (Tidwell, 1997), (Coram and Lee, 1998), (Welie, 1999), (Horton, 1994), and (Nielsen, 1999);
- Si Alhir's (2003) Understanding the Model Driven Architecture (MDA), Methods & Tools';
- Paternò's (2000) Model-Based Design and Evaluation of Interactive Systems;
- Souchon's et al. (2002) Task Modeling in Multiple Contexts of Use;
- Msheik's (2004) Compositional Structured Components Model: Handling Selective Functional Composition;
- Puerta's (1993) Modeling Tasks with Mechanisms.

1.3 Architectures

Software architecture has emerged as an important sub-discipline in software engineering, particularly in large system development. To clarify the notion of architecture in this research project, the adopted definition given by Bass et al. is as follows: "*The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them*" (Bass et al., 2003).

1.3.1 N-tiers

POMA is to be based on the specification of software architecture for interactive systems, where software architecture means: "*the description of the subsystems and components of a software system and the relationships between them, typically represented in different views to show the relevant functional and non functional properties*" (Buschmann et al., 1996). This definition introduces both the main architecture elements (subsystems, components, connectors), how to represent them (by means of a set of different views) and what they actually reflect (both functional and non functional requirements).

The basic architecture that is considered as a starting point is MVC (Goldberg, 1984) pattern implementation in Java language. Such an implementation exploits the Observer Interface and Observable classes. The MVC is a 3-tier architecture with a classic design pattern often used by an interactive systems architecture that needs the ability to maintain multiple views of the same data. The MVC pattern depends on a clean separation of objects into one of three components or levels as follows:

- **Model:** for maintaining data;
- **View:** for displaying all or a portion of the data;
- **Controller:** for handling events that affect the model or view(s).

This MVC (Goldberg, 1984) pattern is illustrated in the UML class diagram in Figure 1.3.

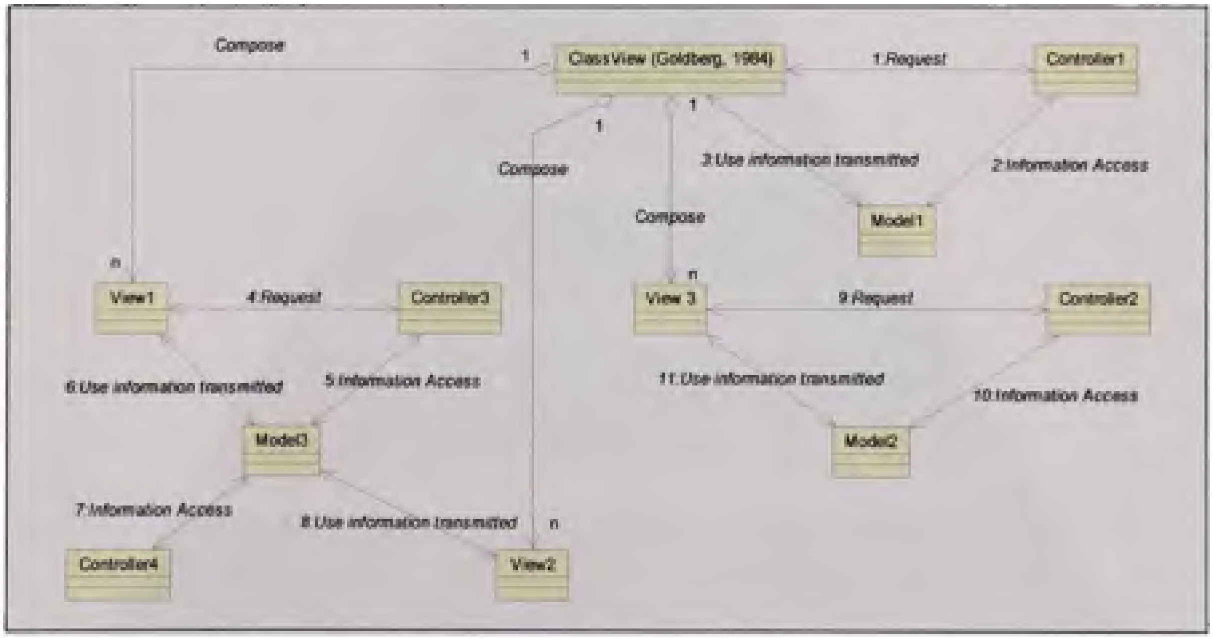

Compose
1
ClassView (Goldberg, 1984)
1.Request
Controller1
1
1
Compose
3:Use information transmitted
2:Information Access
Compose
Model1
n
n
View1
4:Request
Controller3
View 3
9:Request
Controller2
6:Use information transmitted
5:Information Access
11:Use information transmitted
10:Information Access
Model3
Model2
7:Information Access
8:Use information transmitted
n
Controller4
View2

Figure 1.3 Class Diagram of Model-View-Controller architectural pattern.

The Java programming language provides support for the Model-View-Controller architecture with two classes. Such implementation exploits the Observer Interface and the Observable classes.

The main advantage is the decoupling between the views and the models. However, the views are tightly coupled. The *Command Action* pattern is suggested by (Gamma et al., 1995) to ensure the separation between the views and the controller.

The Java Sun team proposes a five-tier architecture to model Core J2EE Patterns Architecture (Sun Microsystems, 2002a) (Java Based Architecture) illustrated on the UML class diagram (see Figure 1.5). This architecture is divided into five levels of patterns (Browser, Presentation, Business logic, Midleware, and Persistence) in Figure 1.4.

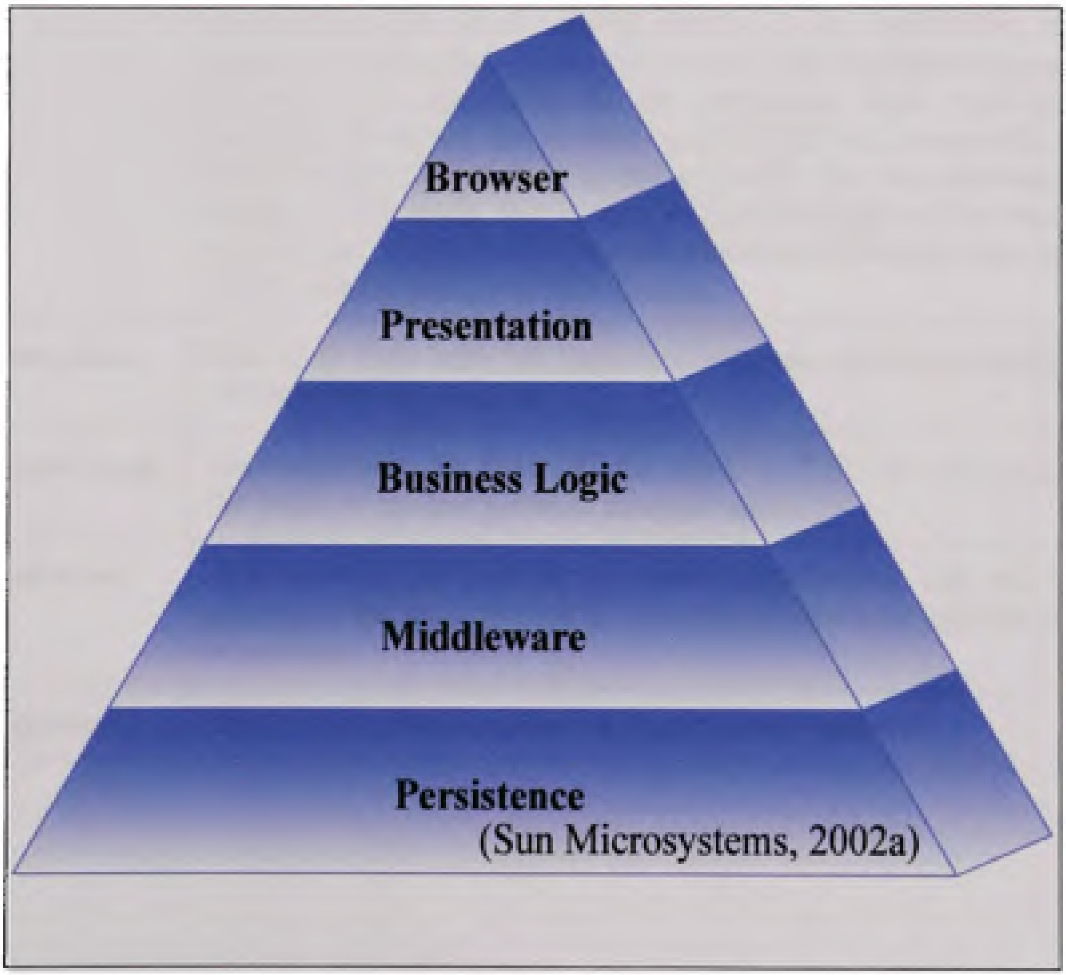

Browser
Presentation
Business Logic
Middleware
Persistence
(Sun Microsystems, 2002a)

Figure 1.4 Core J2EE 5-tier architectural level of patterns.

Table 1.1 gives a description of each of the architectural levels of patterns of (Sun Microsystems, 2002a).

Table 1.1

Description of architectural levels of patterns

Level of Patterns	Description (Sun Microsystems, 2002a)
Browser	This level is very often non-representative of an architecture that contains an applicative part commonly called "Tests of first level". The test of first level consists mainly in the verification of the contents of the capture forms. The tests of first level make it possible to assure that the whole of the mandatory fields were indeed indicated. However, this series of tests MUST form part of the level of presentation. Indeed, it is not excluded that the end-user may decide to deactivate the JavaScript functionalities of his browser. Another use of this level is the representation of the dynamic pages, among others, with a DHTML format.
Presentation	This level deals with the logic of navigation and often implements JSP/Servlets technologies.
Business Logic	Implemented in the form of Java Beans or EJB, this level contains the whole of the treatments of an application.
Middleware	This level of the architecture covers connections with the other patterns of the same level or the composed patterns of different levels of patterns.
Persistence	This level is often composed of one or several patterns.

An example of a UML class diagram of J2EE Patterns is presented in Figure 1.5.

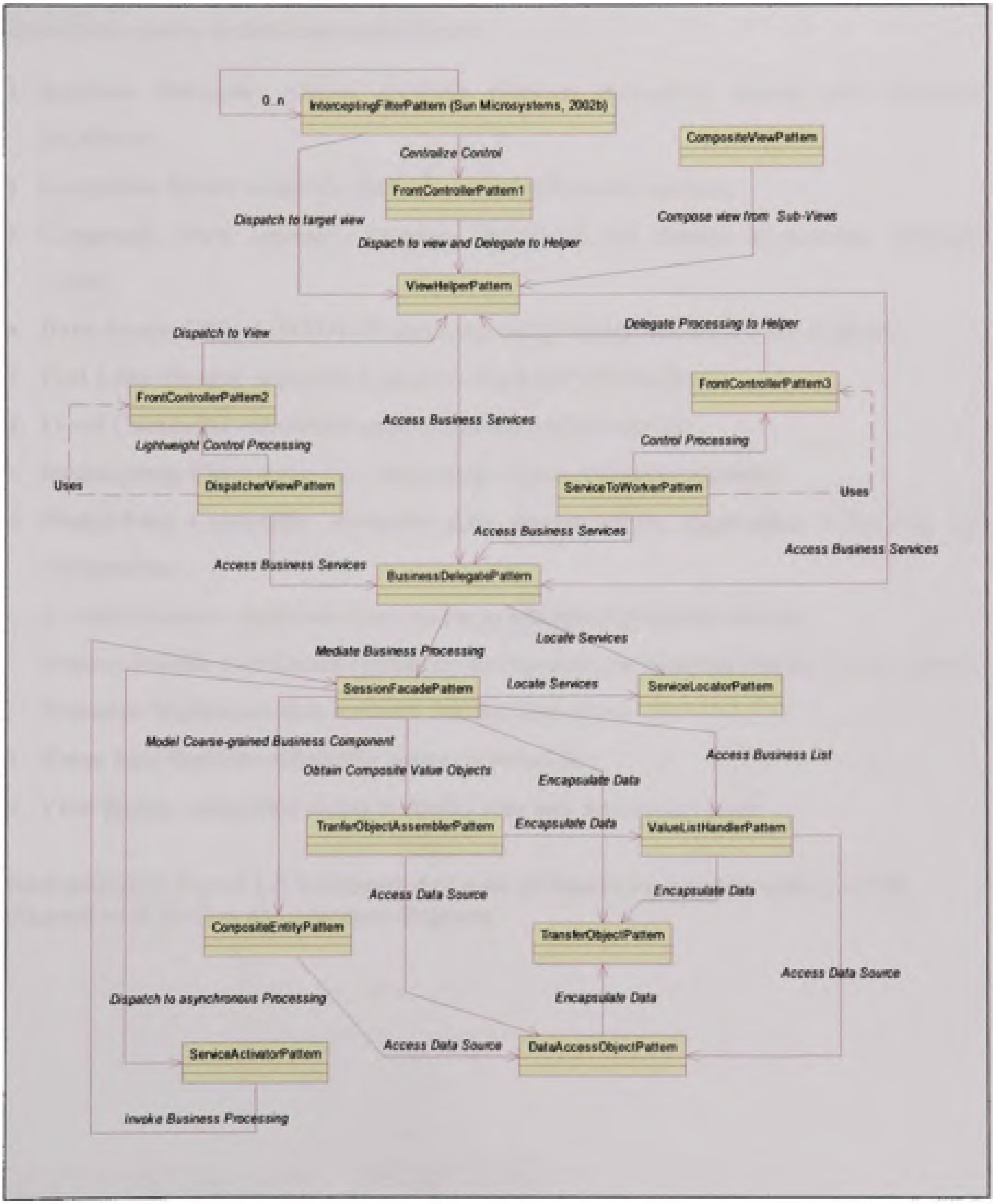

0..n
InterceptingFilterPattern (Sun Microsystems, 2002b)
CompositeViewPattern
Centralize Control
FrontControllerPattern1
Dispatch to target view
Compose view from Sub-Views
Dispach to view and Delegate to Helper
ViewHelperPattern
Dispatch to View
Delegate Processing to Helper
FrontControllerPattern3
FrontControllerPattern2
Access Business Services
Lightweight Control Processing
Control Processing
Uses
DispatcherViewPattern
ServiceToWorkerPattern
Uses
Access Business Services
Access Business Services
Access Business Services
BusinessDelegatePattern
Locate Services
Mediate Business Processing
SessionFacadePattern
Locate Services
ServiceLocatorPattern
Model Coarse-grained Business Component
Access Business List
Obtain Composite Value Objects
Encapsulate Data
TranferObjectAssemblerPattern
Encapsulate Data
ValueListHandlerPattern
Access Data Source
Encapsulate Data
CompositeEntityPattern
TransferObjectPattern
Access Data Source
Dispatch to asynchronous Processing
Encapsulate Data
Access Data Source
DataAccessObjectPattern
ServiceActivatorPattern
Invoke Business Processing

Figure 1.5 UML Class Diagram of Core J2EE Patterns Architecture.

Each pattern of this architecture is described below.

A description for each pattern of Core J2EE (Sun Microsystems, 2002c) pattern-oriented interactive system architecture is as follows:

- **Business Delegate** reduces coupling between interactive system and Enterprise JavaBeans;
- **Composite Entity** models a network of related business entities;
- **Composite View** separately manages the layout and content of multiple composed views;
- **Data Access Object (DAO)** abstracts and encapsulates data access mechanisms;
- **Fast Lane Reader** improves read performance of tabular data;
- **Front Controller** centralizes application request processing;
- **Intercepting Filter** treats pre- and post-process application requests;
- **Model-View-Controller** decouples data representation, application behaviour, and presentation;
- **Service Locator** simplifies client access to enterprise business services;
- **Session Facade** coordinates operations among multiple business objects in a workflow;
- **Transfer Object** transfers business data between tiers;
- **Value List Handler** efficiently iterates a virtual list;
- **View Helper** simplifies access to model state and data access logic.

Each pattern in Figure 1.5 is composed of a set of classes represented within a UML diagram such as class and sequence diagrams.

However, an MVC pattern has certain weaknesses including:

1. This pattern does not encourage the designer to consider other aspects of the dialogue which are very important for the user, such as the help with or the management of errors;
2. This pattern does not facilitate the use of the constraints for the design and the description of the interface, whereas they are of great importance to the designer (Booch et al., 1999), (Myers, 1986), (Myers, 1989a), (Myers, 1989b), (Myers, 1989c) and (Meyer, 1990);
3. The models of architectures are poorly located compared to the life cycle of the user interface. In particular, the difficulties relate to the passage of the analysis of the problem (analyzes users' needs), expressed generally in terms of tasks and sequences of interaction with the concepts proposed by these architectures (agents, components of presentation, and components of dialogue).

Thus, the proposed architecture is designed to be generic and comprised of six levels of patterns. This is based on the Zachman (1987) framework, which also has six categories of patterns. Zachman (1987) and Sowa et al. (1992) proposed a Multi-tiered architecture. Zachman (1987) proposed an Enterprise Architecture schema in which he depicted two distinct dimensions in a matrix:

1. The column classifies answers to the interrogatives: What (Data), How (Function), Where (Network), Who (People), When (Time) and Why (Motivation);
2. The rows classify the audience perspectives of the scope, owner, designers, builder, trades and functioning enterprise. This gives 36 cells which uniquely classify portions of the enterprise.

The columns in the Zachman (1987) framework and Sowa et al. (1992) represent different areas of interest for each perspective. The columns describe the dimensions of the systems development effort. The column descriptions of the Zachman (1987) framework are as follows:

- **WHAT** (Data): Each of the rows in this column address understanding of and dealing with an enterprise's data;
- **HOW** (Function): The rows in the function column describe the process of translating the mission of the enterprise into successively more detailed definitions of its operations;
- **WHERE** (Network): This column is concerned with the geographical distribution of the enterprise's activities;
- **WHO** (People): This column describes who is involved in the business and in the introduction of new technology;
- **WHEN** (Time): This column describes the effects of time on the enterprise;
- **WHY** (Motivation): As described by Mr. Zachman, this column is concerned with the translation of business goals and strategies into specific ends and means.

Considering this Zachman (1987) framework, a pattern-oriented interactive system six-tier architecture is proposed for POMA. The matrix classification of Zachman is on the columns interrogatives; and the rows are the six architectural levels and categories of patterns defined in Table 1.2.

Table 1.2

Pattern-Oriented generic classification schema for POMA Architecture

Architectural level and categories of patterns	WHAT (Data)	HOW (Function)	WHERE (Network)	WHO (People)	WHEN (Time)	WHY (Motivation)
Information	✓	✓	✓	✓	✓	✓
Interoperability		✓	✓		✓	
Visualization	✓	✓		✓		✓
Navigation	✓	✓		✓		✓
Interaction	✓	✓		✓		✓
Presentation	✓	✓		✓		✓

1.3.2 Pattern-Oriented Design (POD)

1.3.2.1 Overview

As the complexity of software systems increases, the software engineering research community looks for new approaches to facilitate the development of software applications. Design patterns and development architectures are among these promising approaches. Design reuse has emerged with the premise that coding is not the most difficult part of building software. The design patterns allow the reuse benefits early in the development lifecycle. To reap the benefits of deploying these proven design solutions, Pattern-Oriented Design (POD) needs to define design composition techniques to construct applications using patterns. Design models should be developed to support these techniques. Several catalogues of design patterns (Yacoub and Ammar, 2003) have emerged with design patterns that can be used in the design of various application domains from real-time embedded systems applications to large distributed systems.

The medium (e.g. natural language or narrative text) generally used to document patterns, coupled with a lack of tool support, compromises the potential use of patterns. These preliminary observations motivated the research community to investigate a systematic approach for incorporating patterns to achieve design solutions (Javahery and Seffah, 2002). The research results support a pattern-based development among software developers who are unfamiliar with HCI design and usability engineering techniques. POD involves transferring the knowledge gained by experts to software engineers through a systematic approach facilitated by tool support. POD motivation helps novice designers apply patterns correctly and efficiently. A tool to support the pattern-oriented design should enhance the pattern user's understandability, decrease the complexity of a pattern and eliminate terminological ambiguity. At the same time, the pattern language should be put into practice in a real context of use, which is difficult when making pattern languages a cost-effective vehicle for gathering and disseminating the best design practices among software and engineering teams.

POD can help with decoupling the different aspects of interactive system architectures and isolate platform specifics from remaining concerns that are common to all platforms.

As with other multi-tiered architectures such as client-server architectures, POD proposes a common information repository (Yacoub and Ammar, 2003), (Zachman, 1987), and (Sowa et al., 1992) which is at the core of multi-layer architectures. Services should be accessed strictly through an adaptable presentation layer, which provides decoupling of the data from the device-specific interfaces. In this way, developers need only worry about the standardized middleware interface (middleware is software which supports communication between the tier components of an interactive system, two or several interactive systems and shared services) rather than having to worry about the multitude of toolkits put forth by database repository manufacturers. Segmenting the architecture and reducing coupling to stringent specifications allow designers to quickly understand how changes made to a particular component affect the remaining interactive system (Yacoub and Ammar, 2003), (Zachman, 1987), and (Sowa et al., 1992).

1.3.2.2 Composition techniques

However, the development of interactive systems using design patterns as design components requires a careful look at composition techniques. Several techniques have been proposed for composition. For example, Yacoub and Ammar (2003) proposed two composition techniques categorized and illustrated in Figure 1.6 as:

- **Behavioral composition techniques** that are based on object interaction specifications to show how instantiations of patterns can be composed.
- **Structural composition techniques** which are based on the static architectural specifications of composed instantiated patterns using class diagrams.

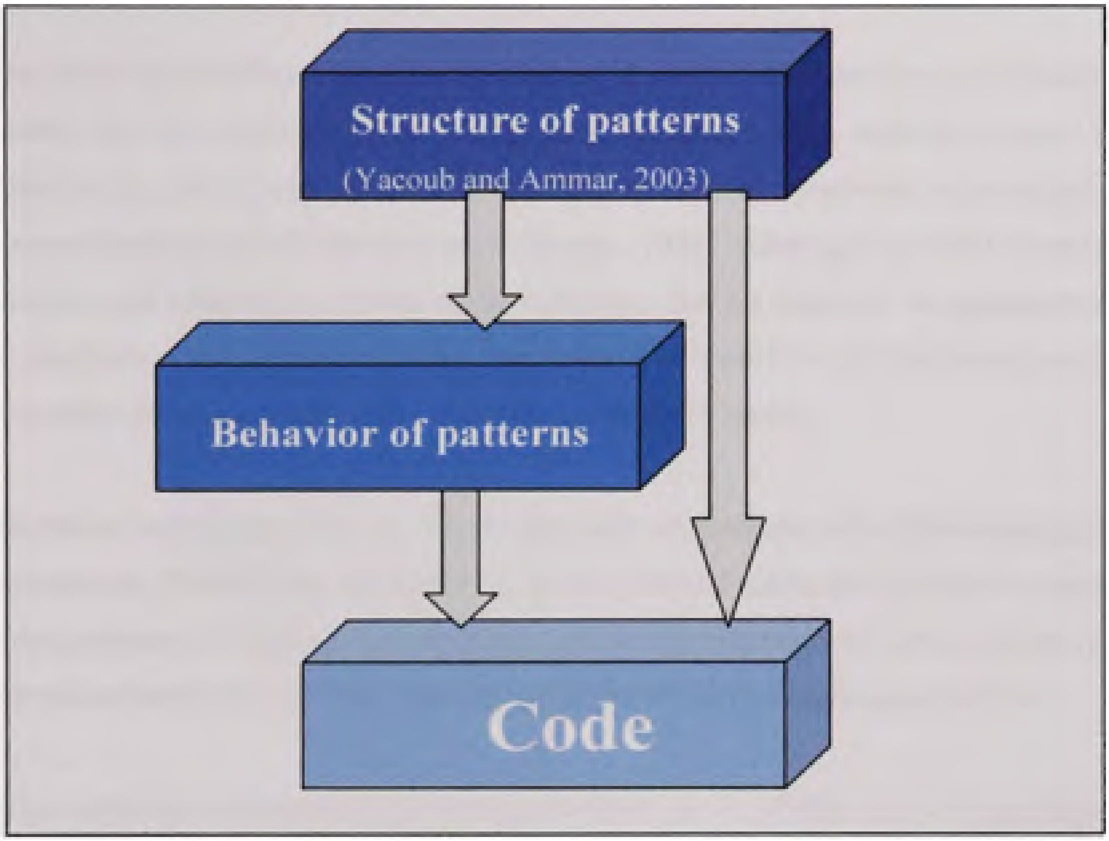


Figure 1.6 Patterns-Oriented Design architecture.

A hybrid technique showing both structural composition and behavioral composition may evolve as a more comprehensive approach for specifying how instantiated patterns can be composed.

The POD composition techniques describe how pattern instances can be composed together as building blocks to develop composite patterns, *OO* applications, or *OO* architectures, relating to the difficulty of composing interactive system at the design level. Understanding the relationships between individual patterns is a good practice but does not solve the issues related to pattern composition.

A. Behavioral Composition Techniques

Behavioural composition techniques are concerned with object interaction specifications as elements that play multiple roles, where each role is part of a separate pattern. These techniques are also known in the *OO* literature as interaction-oriented or responsibility-driven composition (Wirfs-Brock and Wilkerson, 1989). Although the POD composition technique uses notation and composition techniques that are based on the pattern structure (i.e., its class model), (Yacoub and Ammar, 2003) find it useful to be familiar with existing composition techniques that utilize the pattern's behavior model.

Behavioural techniques make up the instantiations of patterns, with both advantages and disadvantages. Formalizing the behavior specification of individual patterns is important for the purpose of clarifying their semantics and facilitating their utilization by any pattern composition technique. Several authors have proposed various techniques, such as:

1. The technique presented by (Henderson-Sellers et al., 1996) on role modeling and synthesis using the *OO* role analysis method;
2. The works of Dirk Riehle (1997) presented at the *OOPSLA* conference in 1997. This technique in (Henderson-Sellers et al., 1996) and (Riehle, 1997) applies the concepts of role models suggested by Reenskaug to pattern composition;
3. The technique called "the superimposition" proposed by Jan Bosch (1998), which uses design patterns and frameworks as architectural fragments and merges roles and components to produce applications;
4. Another technique, a three-layer "role/type/class", is proposed and developed by Lauder and Kent (1998), which takes a visual specification technique to describe design patterns.

B. Structural Composition Techniques

Structural composition techniques build a design by gluing together pattern structures that are modeled as class diagrams using static architectural specifications. Structural composition focuses more on the actual realization of the design rather than abstraction, using different types of models, such as role models. Behavioral composition techniques, such as roles (Henderson-Sellers et al., 1996), (Riehle, 1997) and (Kristensen and Østerbye, 1996) leaving several choices to the designer with fewer insights on how to continue to the class design phase.

Techniques that consider both structural and behavioral views could be complex and difficult to use. Therefore, the POD architecture advocates a structural composition technique with pattern class diagrams (Henderson-Sellers et al., 1996), (Riehle, 1997) and (Kristensen and Østerbye, 1996). Constructional design patterns in which a pattern interface can be clearly specified lend themselves to a structural composition technique (Henderson-Sellers et al., 1996), (Riehle, 1997) and (Kristensen and Østerbye, 1996).

(Yacoub and Ammar, 2003) discuss several structural composition techniques and contrast these techniques with a proposed POD architecture. One approach for pattern-oriented design is proposed by Ram, Anantha, and Guruprasad (1997). In contrast to the top-down approach, this approach describes a bottom-up process to design software using design patterns. This approach shows how related patterns can be selected but does not clearly show how patterns can be composed. Nevertheless, this is an example of previous attempts in the literature to develop a systematic process for pattern-oriented software development.

1.3.2.3 Other techniques

A number of other techniques have been suggested. Keller and Schauer's (1998) "Design Components: Towards Software Composition at the Design Level", Wills and D'Souza's (1996) "Component and Framework-based Development", Msheik, Abran and Lefebvre's

(2004) "Compositional Structured Components Model: Handling Selective Functional", Clarke and Walker's (2001) "Composition Patterns. An Approach to Designing Reusable Aspects" and Clark's (2000) "Composing Design Models: An Extension to the UML", and Larsen's (1999) "Designing Component-Based framework Using Patterns in the UML".

1.3.3 Pattern Supported Approach (PSA)

The "Pattern Supported Approach" (PSA) addresses patterns not only during the design phase, but also during the entire software development process. PSA (Granlund et al., 2001) aims to support early system definition and conceptual design through the use of patterns. In particular, patterns have been used to describe business domains, processes, and tasks to aid early system definition and conceptual design. The main idea of PSA is that patterns can be documented according to the development lifecycle. During system definition and task analysis, depending on the context of use, one can decide which patterns are appropriate for the design phase. In contrast to POD, the concept of linking patterns together to result in a design is not tackled in this architecture (Sinnig, 2004).

The Pattern-Supported Approach (PSA) to the user interface design process suggests a wider scope for the use of patterns by looking at the overall design process. Since the usability of a system emerges as the product of the user, the task and the context of use, PSA integrates this knowledge into most of its patterns, dividing the forces in the pattern description correspondingly (i.e., describing Task and Subtask, User, and Context forces). PSA provides a double-linked chain of patterns (parts of an emerging pattern language) that *support* each step of the design process (Granlund et al., 2001).

PSA proposed architecture highlights another important aspect of Pattern-Oriented Design, that of *pattern combinations*. By combining different patterns, developers can use pattern relationships, combining them, in order to produce an effective design solution (Sinnig, 2004). Most of the work on patterns has focused on screen design issues. PSA addresses patterns not only at the design phase, but also *before* design (Figure 1.7).

For example, *task* patterns point to *Structure and Navigation Patterns*, which in turn point to *GUI* *Design Patterns*, and vice-versa. These patterns offer a way to capture and communicate knowledge from previous designs (including the knowledge from system definition, task/user analysis and structure & navigation design). Given a mature language of patterns belonging to the described classes, the PSA approach provides an entry point to this pattern language and suggests (without restricting the pattern usage) a chain of appropriate patterns at different levels of analysis and design (Granlund et al., 2001).

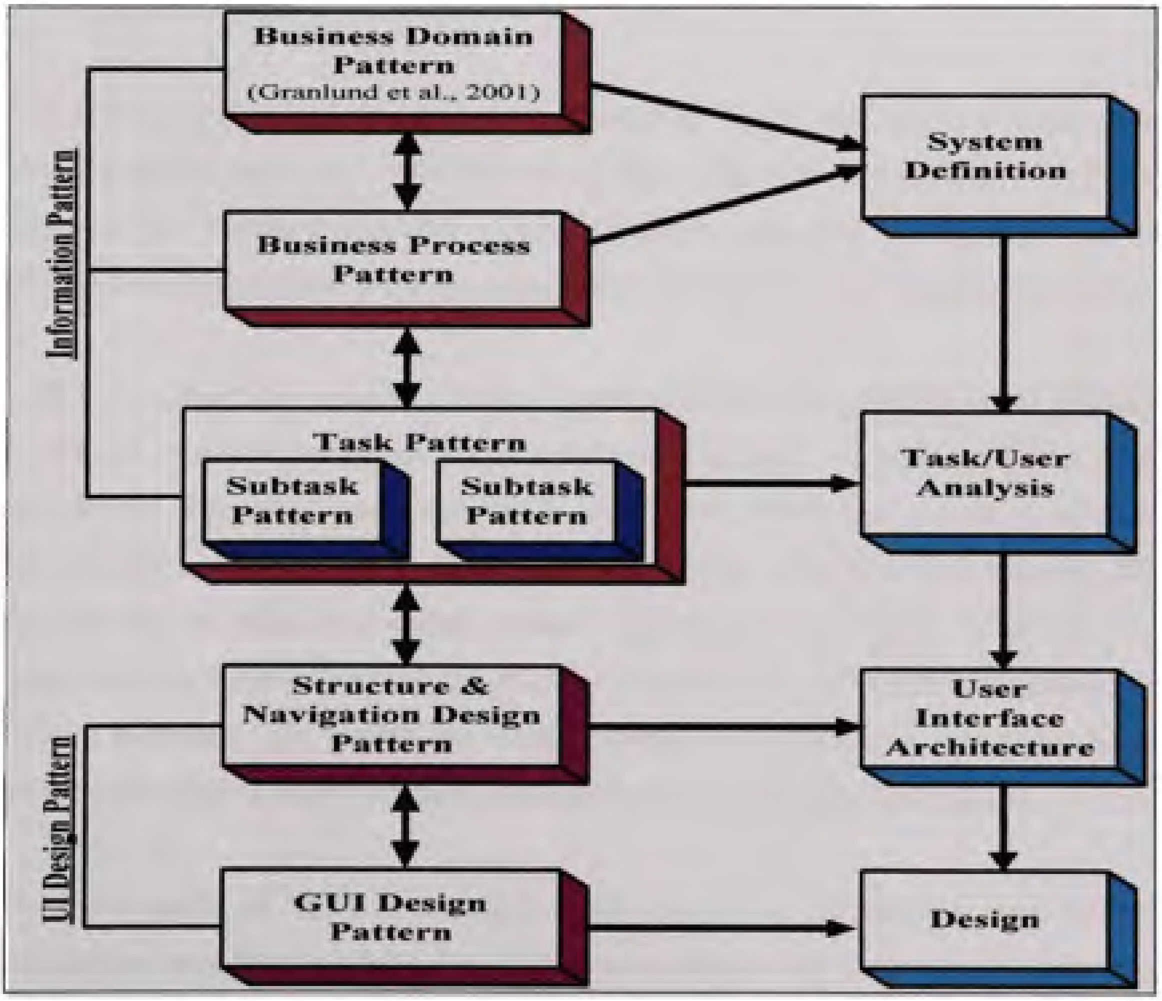

Business Domain Pattern
(Granlund et al., 2001)
Business Process Pattern
Task Pattern
Subtask Pattern
Subtask Pattern
Structure & Navigation Design Pattern
GUI Design Pattern
System Definition
Task/User Analysis
User Interface Architecture
Design
Information Pattern
UI Design Pattern

Figure 1.7 The PSA architecture with the relationships between PSA patterns.

1.3.4 Model-Driven Architecture (MDA)

Models are commonly used to represent flexible complex systems. The models can be viewed at many levels of abstraction and complementary model views can be combined to give a view of a system better than a single model alone. Meservy and Fenstermacher (2005) claim that many software development experts have long advocated using models to understand the problem that an interactive system seeks to address; yet development teams commonly employ models only in the early stages of modeling. Often, once construction begins, the teams leave these models behind and never update them to reflect their changes in a project.

Most software developers would agree that modeling should play a role in every project (Meservy and Fensternacher, 2005). However, there is no clear consensus on what that role should be, how developers should integrate modeling with other development activities, and who should participate in the modeling process (Meservy and Fensternacher, 2005).

In 2001, the Object Management Group introduced Model-Driven Architecture (MDA) as an approach to system specification and interoperability based on the use of formal models (i.e., definite and formalized models) (Sinnig, 2004), (OMG group, 2005), (D'Souza, 2001), (Soley and OMG group, 2000), (Mukerji, 2001) and (Miller and Mukerji, 2003). The main idea of MDA is to specify business logic in the form of abstract models. These models are then mapped (partly automatically) according to a set of transformation rules to different platforms. The models are usually described by UML in a formalized manner which can be used as inputs for tools which perform the transformation process.

The main benefit of MDA is the clear separation of the fundamental logic behind a specification from the specifics of the particular middleware that implements it. The MDA approach distinguishes between the specifications of the operation of a system and the details of the way that the system uses the capabilities of its platform. This architectural separation of concerns constitutes the basic foundation of MDA in order to reach three main goals: portability, interoperability and reusability (Sinnig, 2004), (OMG group, 2005), (Soley and OMG group, 2000) and (Miller and Mukerji, 2003).

The MDA architecture is comprised of three main steps:

- Specifying the system independently from the platform that supports it;
- Specifying target platforms;
- Transforming the system specification into a specification for a particular platform.

Specifying the system:

In this step, a *platform independent model* (PIM) is established. Usually a formalized UML notation is used to specify the PIM which describes the system, but does not show details of its use or its platform. A PIM exhibits a specified type of platform independence to be suitable for use with a number of different platforms of similar type.

Specifying the platform:

In this step, a *platform model* provides a set of technical concepts representing the different kinds of parts that make up a platform and the services provided by that platform. A platform model also provides, for use in a platform specific model (PSM), concepts representing the different kinds of elements needed in specifying the use of the platform by an interactive system. The architect will then choose a platform (or several) that enables implementation of the interactive system with the desired architectural qualities.

Transforming the system specification into a specification for a particular platform:

In this step, the *platform independent model* (PIM) will be transformed into a platform specific model (PSM) according to various mapping rules. In particular, MDA mapping provides specifications for the transformation of a PIM into a PSM for a particular platform. The platform model will determine the nature of the mapping. A mapping may also include templates, which are parameterized models that specify particular kinds of transformations. These templates are like design patterns but may include much more detailed specifications to guide the transformation. Templates can be used in rules for transforming a pattern of model elements to model-type mapping into another pattern of model elements.

A *platform specific model* is a view of an interactive system from the platform specific viewpoint. A PSM combines the specifications in the PIM with details that specify how that system uses a particular type of platform. A PSM may provide more or fewer details, depending on their purpose. Eventually, if the PSM provides all the information needed to construct a system and to put it into operation, it may be used for the implementation of the interactive system.

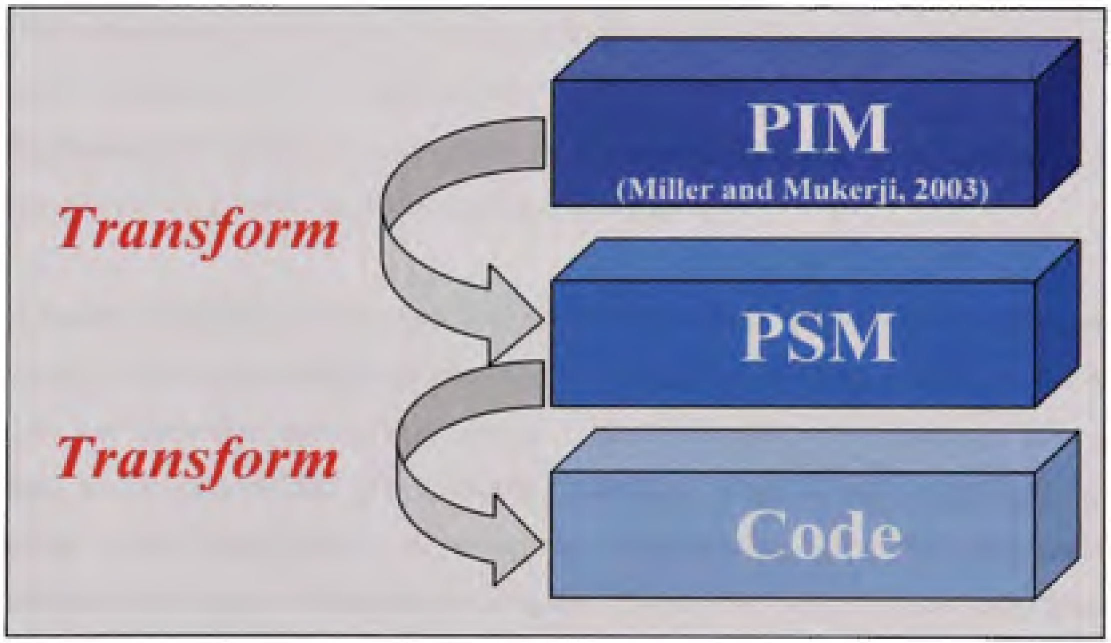


Figure 1.8 Transformations of MDA architecture.

Figure 1.8 represents a transformation from a PIM to a PSM and eventually to the implementation code of the interactive system.

Briefly, MDA makes a sharp distinction:

- The business model (the Computation-Independent Model, or CIM), sometimes called a domain model;
- The domain model in a specific technology context (PIM);
- A model that is tied to the domain and uses a platform-specific code (PSM).

There are two other steps that can be integrated into MDA process development:

- **Capturing requirements in a CIM**. The Computation-Independent Model captures the domain without reference to a particular system implementation or technology. The CIM would remain the same even if the systems were implemented mechanically rather than in computer software, for example.
- **Deploying the system in a specific environment**. Here, the goal is to deploy the system in several specific platforms and environments.

Interoperability had been based mostly on CORBA standards and services. Heterogeneous interactive systems inter-operate at the level of standard component interfaces. The MDA process, on the other hand, places formal system models (i.e. defined and formalized system) at the core of the interoperability problem. What is most significant in this approach is the independence of the system specifications from the implementation technology or platform. As illustrated in Figure 1.9, the PIM exists independently from any implementation of the model and has mappings to many possible platform infrastructures such as CORBA PSM, EJB JAVA PSM, and SOAP PSM. After establishing the PSM, this model must be implemented on the specific target platform (Sinnig, 2004), (D'Souza, 2001) and (Miller and Mukerji, 2003).

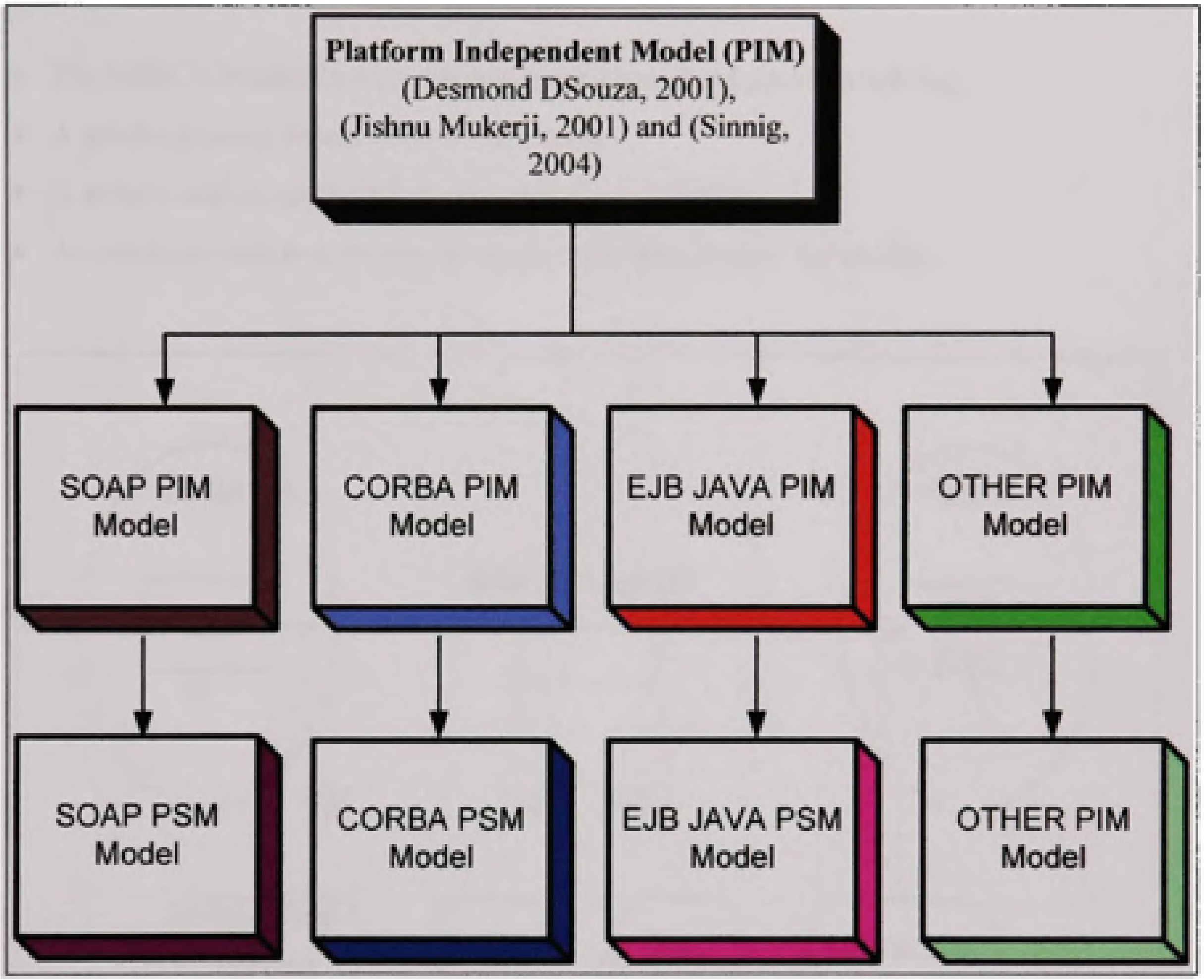

Platform Independent Model (PIM)
(Desmond DSouza, 2001),
(Jishnu Mukerji, 2001) and (Sinnig, 2004)
SOAP PIM Model
CORBA PIM Model
EJB JAVA PIM Model
OTHER PIM Model
SOAP PSM Model
CORBA PSM Model
EJB JAVA PSM Model
OTHER PIM Model

Figure 1.9 PIM, PSM and Implementation.

Foundation

Figure 1.10 shows the foundational concepts that generally constitute an MDA:

- The MDA's Model-Driven Approach corresponds to problem solving;
- A platform corresponds to an environment;
- A system and its applications correspond to a solution;
- An implementation corresponds to an implementation of the models.

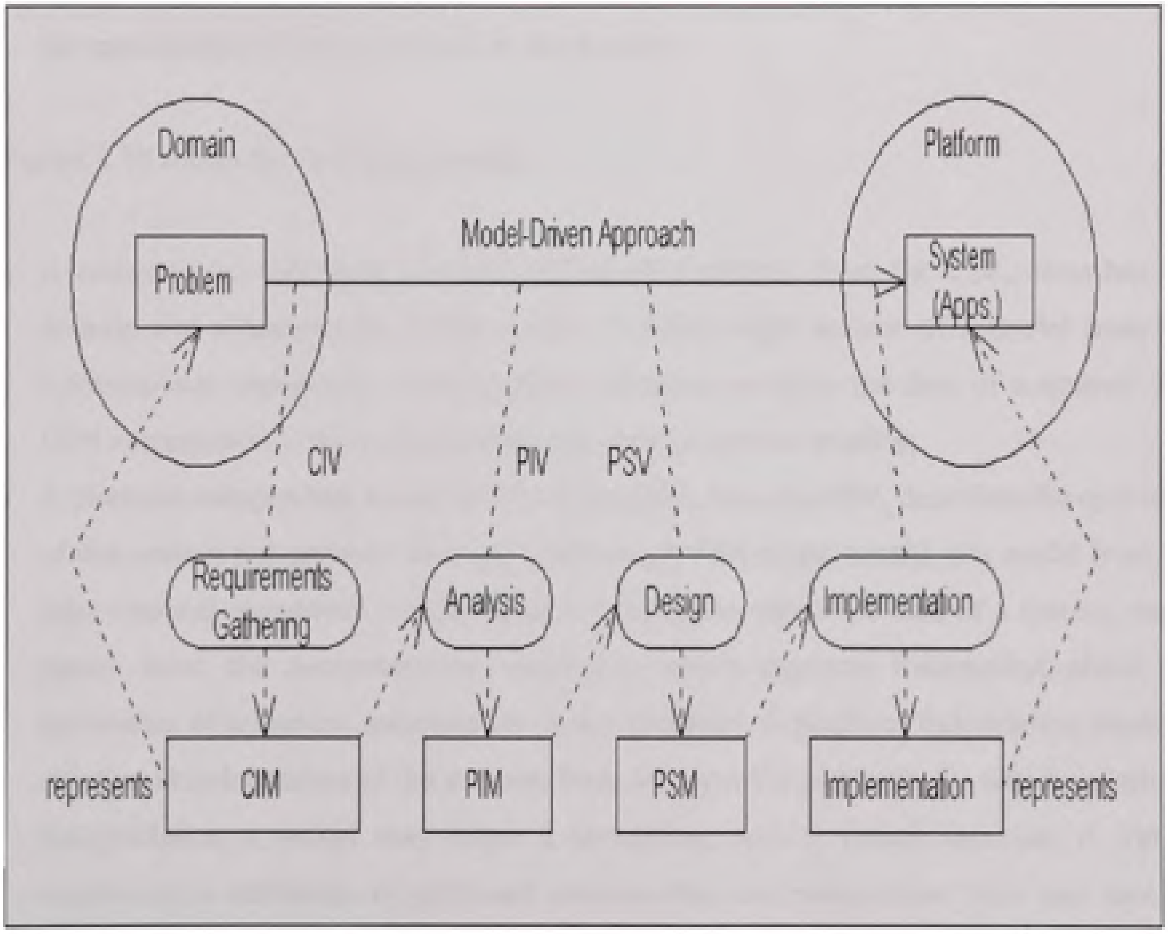

Domain
Platform
Model-Driven Approach
Problem
System (Apps.)
CIV
PIV
PSV
Requirements Gathering
Analysis
Design
Implementation
represents
CIM
PIM
PSM
Implementation
represents

Figure 1.10 Foundational Concepts of the MDA.
(Extracted from architecture of (Si Alhir, 2003))

Figure 1.10 shows the following viewpoints:

- A computation independent viewpoint (CIV) focuses on the requirements of a system and its environment. The CIV corresponds to the conceptualization perspective;
- A platform independent viewpoint (PIV) focuses on the operation of a system independent from any platform and does not change from one platform to another. The PIV corresponds to the specification of analysis activities and model;
- A platform specific viewpoint (PSV) focuses on the operation of a system based on a specific platform and changes form one platform to another. The PSV corresponds to the specification of design activities and models.

Figure 1.10 shows the following models:

- A computation independent model (CIM) of a system, from the CIV, describes the domain and requirements of the system. A CIM might consist of a model from the informational viewpoint, which captures information about the data of a system. The CIM corresponds to the conceptualization of requirements models;
- A platform independent model (PIM) of a system, from the PIV, describes the operation of that system independent from any platform. A PIM might consist of a model from the informational viewpoint, which captures information about the data of a system, and a model from the computational viewpoint, which captures information about the processing of a system, independent of any platform. A platform independent model is one that is independent of the features from any specific platform. To achieve platform independence, a model may target a technology-neutral virtual machine. A virtual machine is a collection of parts and services that are independent from any specific platform and may be realized on multiple specific platforms, but the virtual machine remains independent and unaffected by any underlying platform. The PIM corresponds to the specification of an analysis model;

- A platform specific model (PSM) of a system, from the PSV, describes the operation of the system as it uses one or more specific platforms. A PSM might consist of a model from the informational viewpoint, which captures information about the data of a system, and a model from the computational viewpoint, which captures information about the processing of a system, based on a specific platform. As a PSM targets a specific platform, it uses the features of the specific platform specified by a platform model. The PSM corresponds to the specification of a design model.

1.4 Why Combine Patterns and Models?

In an attempt to segment the different aspects of interactive system architecture and isolate specific platforms from remaining issues, the industry of interactive systems has adopted a layered approach. As with other multi-tiered architectures such as client-server architecture, a common information repository is at the core of the architecture. The repository is accessed strictly through this layer, which in addition to the functions listed, also provides decoupling of the data from the device specific interfaces. In this way, device interactive system developers need only worry about the standardized middleware interface rather than having to concern themselves with the multitude of APIs put forth by database repository manufacturers.

Segmenting the architecture and reducing coupling to stringent specifications allows designers to understand quickly how changes made to a particular component affect the remaining interactive system since achieving these goals requires a consistent approach in applying both cognitive and social factors to user interface design, and requires independent developers to coordinate their activities.

Interactive systems can also be much more efficient at managing heterogeneous environments. This latter point is critical as more and more systems will need to interact with very different platforms and devices. This diversity results in computing devices that exhibit drastically different capabilities. For example, Personal Digital Assistants (PDAs) use a pen based input mechanism and have average screen sizes in the range of 3 inches. On the other hand, the typical PC uses a full size keyboard, a mouse and has an average screen size of 17 inches. Coping with such drastic variations implies much more than mere layout changes. Pen based input mechanisms are slower than traditional keyboards and are inappropriate for systems such as word processing that require intensive user input. Similarly, the small screens available on many PDAs provide only coarse graphic capabilities and would be ill-suited for photo editing applications.

Another challenge is the heterogeneity in computing platforms ranging from traditional desktop to mobile phone via PDA. This source is a further complication for systems engineering. Certain form factors are better suited to particular contexts. For example, while walking down the street, a user may use his mobile telephone's Internet browser to view a stock quote. However, it is highly unlikely that this same user will review the latest changes made to a document using the same device. Rather, it would seem more logical and definitely more practical to use a full size computer for this task. It would therefore seem that the context of use is determined by a combination of internal and external factors. The internal factors primarily relate to the user's attention while performing a task. In some cases, the user may be focused entirely while, at other times, greatly distracted by other concurrent tasks. As an example of this latter point, a user, while driving a car, operates a PDA to reference a telephone number. External factors are determined to a large extent by the device's physical characteristics. It is not possible to make use of a traditional PC as one walks down the street – a practice quite common with a mobile telephone. The challenge for a system architect is therefore to match the design of a particular device's user interface with the set of constraints imposed by the corresponding context of use.

Finally, many system manufacturers and researchers have issued design guidelines to applications designers (Buschmann et al., 1996). Recently, Palm Inc. has put forth design guidelines to address navigation issues, widget selection, and use of specialized input mechanisms such as handwriting recognition. Macintosh (1992), Microsoft (1995), Sun Microsystems (2001), and IBM (2007) have also published their own usability guidelines to assist developers with programming applications targeted at the Pocket PC/Windows CE platform.

However, these guidelines are different from one platform or device to another. When designing a multi-device application, this can be a source of a number of inconsistencies. The Java "look-and-feel" developed by Sun Microsystems (2001) is a set of cross-platform guidelines that can correct such problems. However, cross-platform guidelines do not take into account the particularities of a specific device, especially platform constraints and capabilities. This can be a source of problems for a user requiring different types of devices to interact with the server side services and information of a system. Furthermore, for a novice designer or a software engineer who is not familiar with this mosaic of guidelines, it is difficult to remember all the design guidelines and their effective use. It is sometimes difficult to make trade-offs among these principles when they come into conflict. The best solution is often made by guessing, or by resorting to other means.

1.5 Summary of chapter

This chapter has provided an overview of the related work. In current practice, there are various architectures based on models and patterns that allow for the design and development of interactive systems. These architectures include:

1. N-Tiers architectures based mainly on architectural levels;
2. Pattern-Oriented Design based (POD) mainly on composition techniques of patterns;
3. Pattern-Supported Approach (PSA) based mainly on categories of patterns;
4. Model-Driven Architecture (MDA) based mainly on PIM and PSM models and on transformation and mapping models.

This chapter has presented various architectures and their characteristics and criteria relevant to this research project (Table 1.3). These architectures are:

- N-tiers architectures such as: MVC 3-tiers, J2EE 5-tiers, and Zachman multi-tiers;
- Pattern-Oriented such as: Pattern-Oriented Design (POD) and Pattern-Supported Approach (PSA);
- Model-Driven such as: Model-Driven Architecture (MDA).

The characteristics and criteria of these architectures quoted above are architectural level, solution, proven concept, relationship, human component, structural technique, behavioral technique, tasks and subtasks and models. Table 1.3 summarizes an assessment of architectures presented in Chapter 1 according to the established characteristics and criteria for defining the research project issues.

Table 1.3

Summary of architectures (N-tiers, POD, PSA, MDA) and their characteristics assessment

Patterns and Architectures	Characteristics and criteria								
	Architectural Level	Solution	Proven concept	Relationship	Human component	Structural technique	Behavioral technique	Tasks and Subtasks	Models
Patterns (is not an architecture)	No	Yes	Yes	Yes	Yes	No	No	No	No
N-tiers	Fully	Partially	Yes	No	No	Yes	Yes	No	Yes
POD	No	Partially	Yes	No	No	Fully	Fully	No	No
PSA	No	Partially	Yes	No	No	No	No	Fully	No
MDA	Yes	Partially	Yes	No	No	No	No	No	Fully

In chapter 2, research issues are introduced that lead to the research statement, the research goal and objectives including the research steps, the research scope and the research methodology.

CHAPTER 2

RESEARCH ISSUES

This chapter describes the methodological aspects of this research. The first section presents the research goals and objectives. The second section deals with the research scope. The third section describes the exploratory research methodology applied to evaluate a Pattern-Oriented and Model-Driven Architecture, called POMA that is based on architectural levels and categories of patterns and on model categorization.

The majority of the architectures studied in chapter 1 such as POD, PSA and MDA are incomplete in the sense where each does not integrate the important concepts of another for the development of interactive systems; see the detail in Table 1.3 presented in section 1.5. For example:

- MDA doesn't take into account the concept of patterns;
- POD and PSA do not take account the concept of models.

In addition, the composition and mapping rules of the patterns and the model transformation rules are not existed.

2.1 Research Goal and Objectives

The goal is to:

> "Define a new architecture to facilitate the development and migration of interactive systems while improving their usability and overall quality".

To pursue to this goal, the research objective is to develop a new architecture which is called, **P**attern-**O**riented and **M**odel-Driven **A**rchitecture (**POMA**). POMA will be based on the concepts of N-tiers architectures (MVC, J2EE, Zachman), Pattern-Oriented Design (POD), the Pattern-Supported Approach (PSA), and Model-Driven Architecture (MDA).

2.2 Research Scope

The following delimits the scope of the proposed architecture:

- The source code generation of interactive systems is not taken into account;
- The architecture is driven by conceptual models;
- Patterns are building blocks in the construction and the transformation of the models;
- A pattern composition is a process of combining architectural levels and categories of patterns to create a platform independent model (PIM) using composition rules that are described in the "Pattern Composition" section of chapter 4;
- Pattern mapping is the process of creating a design of specific models (PSM) for each platform from independent models (PIM) using the mapping rules that are described in the "Patterns Mapping" section of chapter 4;
- A transformation of models is the process of converting one or more models – called source models – to an output model – the target model – of the same system using the transformation rules as described in the "Models Transformation" section of chapter 5.

2.3 Research Methodology

The research methodology (Figure 2.1) designed to attain the research objectives includes the following research steps:

1. Elaborate the background and related work on patterns, models, architectures such as N-tiers architectures, Pattern-Oriented Design (POD), Pattern-Supported Architecture (PSA), and Model-Driven Architecture (MDA) (details in chapter 1);
2. Define the research objectives (details in chapter 2);
3. Propose the POMA architecture (details in chapter 3);
4. Describe the architectural levels and categories of patterns (Navigation patterns, Interaction patterns, Visualization patterns, Presentation patterns, Interoperability patterns, and Information patterns) as well as different relationships between patterns to be used in the POMA architecture. Their relationships are used to combine and to map these categories of patterns to create a pattern-oriented design for interactive systems, and to generate specific implementations suitable to different platforms from the same pattern-oriented design (details in chapter 4);
5. Propose the five categories of models (❶ Domain model, ❷ Task model, ❸ Dialog model, ❹ Presentation model and ❺ Layout model) that resolve some of the challenging problems such as: (a) decoupling the various aspects of interactive systems including business logic, user interface, navigation, and information architecture; (b) isolating platform-specific problems from the concerns common to all interactive systems for architecture, supported by the drafting of an article on these different obtained models (details in the chapter 5);
6. Build an illustrative case study to illustrate and clarify the core ideas of POMA architecture and its practical relevance to interactive systems (details in chapter 6).

Figure 2.1 summarizes the key research methodology steps.

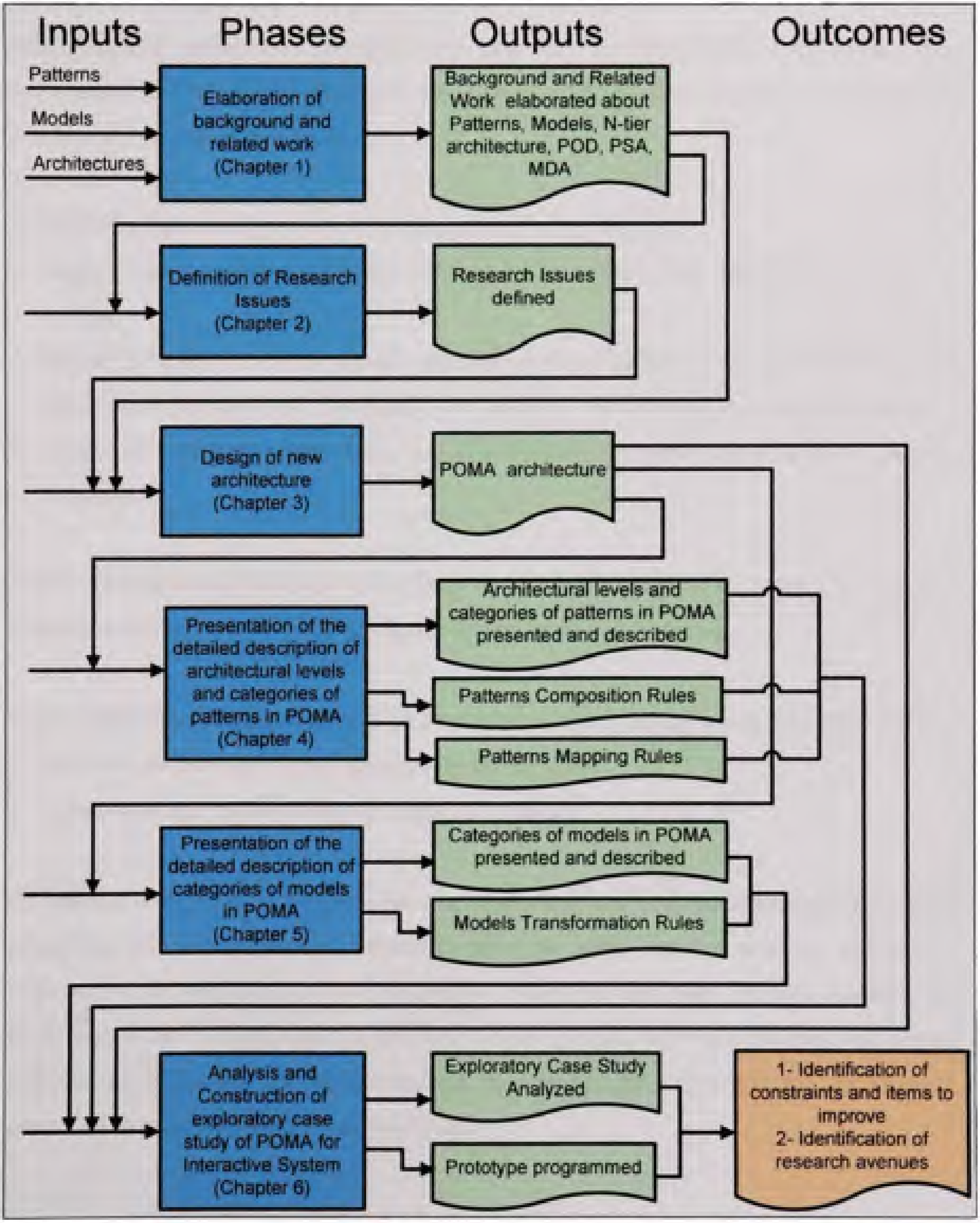

Inputs
Phases
Outputs
Outcomes
Patterns
Models
Architectures
Elaboration of background and related work (Chapter 1)
Background and Related Work elaborated about Patterns, Models, N-tier architecture, POD, PSA, MDA
Definition of Research Issues (Chapter 2)
Research Issues defined
Design of new architecture (Chapter 3)
POMA architecture
Presentation of the detailed description of architectural levels and categories of patterns in POMA (Chapter 4)
Architectural levels and categories of patterns in POMA presented and described
Patterns Composition Rules
Patterns Mapping Rules
Presentation of the detailed description of categories of models in POMA (Chapter 5)
Categories of models in POMA presented and described
Models Transformation Rules
Analysis and Construction of exploratory case study of POMA for Interactive System (Chapter 6)
Exploratory Case Study Analyzed
Prototype programmed
1- Identification of constraints and items to improve
2- Identification of research avenues

Figure 2.1 Methodology Research.

2.4 Summary of chapter

The proposed architecture presents and describes its constituent components, their responsibilities and relationships, and the ways in which several categories of patterns are combined. The architecture seeks to:

1. Facilitate the use of patterns for experts as well as for beginners;
2. Support the automation of approaches to design oriented by patterns and driven by models;
3. Ensure the quality of the produced applications, since pattern-oriented architectures will also have to enable one to encapsulate the quality of attributes and facilitate prediction;
4. Support the communication and reuse of individual expertise regarding good practices of design.

Pattern-Oriented and Model-driven Architecture (POMA) provides on a single level a pool of proven solutions to many recurring design problems which include:

- Decouple the different aspects of an interactive system such as a business logic, the user interface, the navigation and information architecture;
- Isolate platform specifics from remaining concerns.

On another level, POMA illustrates how to combine several individual patterns of different categories into heterogeneous structures and, as such, can be used to facilitate a constructive development of the interactive system architecture. As an example, the patterns that can be used span several levels of abstraction, from information architectural patterns and design patterns for interoperability to navigation, interaction and visualization, presentation and information design patterns.

The originality of the research is to propose a new architecture to integrate all concepts of architectures (N-tiers, POD, PSA and MDA) described in Chapter 2. More importantly, to define composition and mapping rules of patterns and transformation rules of models.

Chapter 3 presents and describes the fundamentals and key concepts, an overview, justifications, and specifications of Pattern-Oriented and Model-driven Architecture (POMA).

CHAPTER 3

POMA: PATTERN-ORIENTED AND MODEL-DRIVEN ARCHITECTURE

This chapter presents the key concepts, an overview, justifications, and specifications of the proposed POMA architecture, which constitutes the development architecture of interactive systems. The first section describes the key concepts. The second gives an overview of POMA. The third section introduces justifications of POMA versus N-tiers architectures, POD, PSA, and MDA. The last section presents specifications for POMA architecture.

3.1 Key concepts of POMA

The five key concepts of POMA are:

- Architectural levels and categories of Patterns (details in chapter 4);
- Models ([POMA.PIM and [POMA.PSM]) (details in chapter 5);
- Pattern composition rules (details in chapter 4);
- Pattern mapping rules: PIM to PSM (details in chapter 4);
- Model transformation rules: PIM to PIM and/or PSM to PSM (details in chapter 5);
- Code generation rules (this level of POMA is not included in this research project).

Figure 3.1 shows the five concepts of POMA and their relationships.

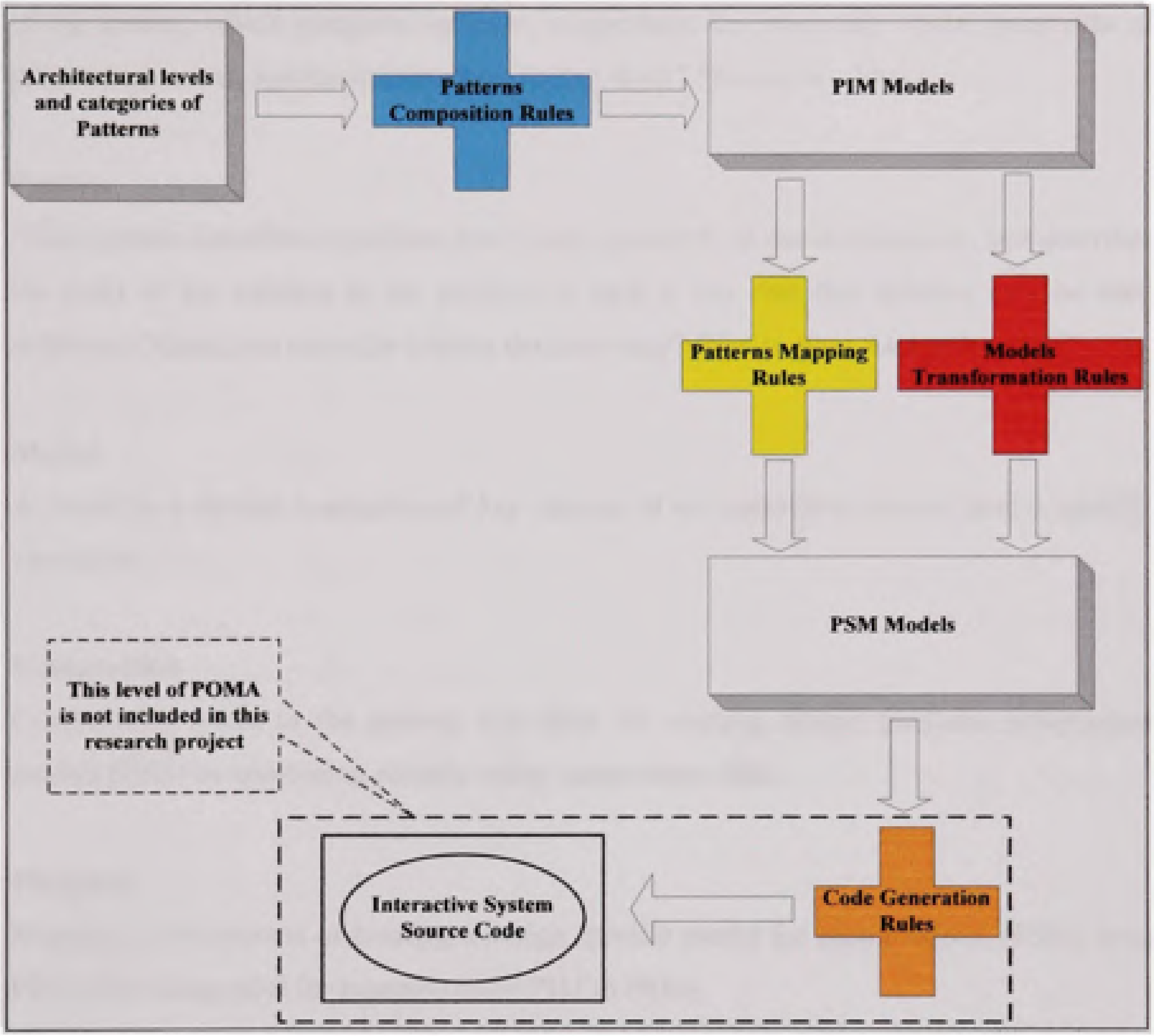

Architectural levels and categories of Patterns
Patterns Composition Rules
PIM Models
Patterns Mapping Rules
Models Transformation Rules
PSM Models
This level of POMA is not included in this research project
Interactive System Source Code
Code Generation Rules

Figure 3.1 Key concepts of POMA.

At this stage, it is interesting to remind some important definitions of each concept used in POMA architecture which is given below.

Architecture

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" (Bass et al., 2003).

Pattern

"Each pattern describes a problem that occurs constantly in the environment, and describes the heart of the solution to the problem in such a way that this solution may be used millions of times, but never do it twice the same way" (Christopher Alexander, 1977).

Model

A model is a formal description of key aspects of an interactive system from a specific viewpoint.

Composition

Composition refers to the process and rules for creating design platform independent models (PIM) by combining patterns using composition rules.

Mapping

Mapping is the process of creating a design specific model for each platform (PSM) from PIM while using rules for mapping (only PIM to PSM).

Transformation

The transformation of models is the process of creating a model from another model using transformation rules (only PIM to PIM and/or PSM to PSM).

3.2 POMA Overview

The proposed POMA architecture (Figure 3.2) for interactive systems development is an architecture comprising five architectural levels of models using six categories of patterns of software architecture.

The POMA architecture (Figure 3.2) includes:

- Six architectural levels and categories of patterns;
- Ten models, five of which are [POMA.PIM] and five others [POMA.PSM];
- Four types of relations used in POMA architecture, which are:
    1. *Composition*: used to combine different patterns to produce a [POMA.PIM] by applying the composition rules;
    2. *Mapping*: used to build a [POMA.PIM] which becomes a [POMA.PSM] by applying the mapping rules ([POMA.PIM] ⇨ [POMA.PSM]);
    3. *Transformation*: used to establish the relationship between two models ([POMA.PIM] ⇨ [POMA.PIM]) and / or ([POMA.PSM] ⇨ [POMA.PSM]) by applying the transformation rules;
    4. *Generation*: used to generate the source code of the whole interactive system by applying the generation code rules. This phase is not taken into account in this research project.

The direction in which to read the POMA architecture in Figure 3.2 is as follows:

- Vertically, concerns the composition of the patterns to produce ten PIM and PSM models;
- Horizontally, concerns the composition and mapping of the patterns to produce five PIM and five PSM models, and the generation of the source code for the whole interactive system (not included in this research).

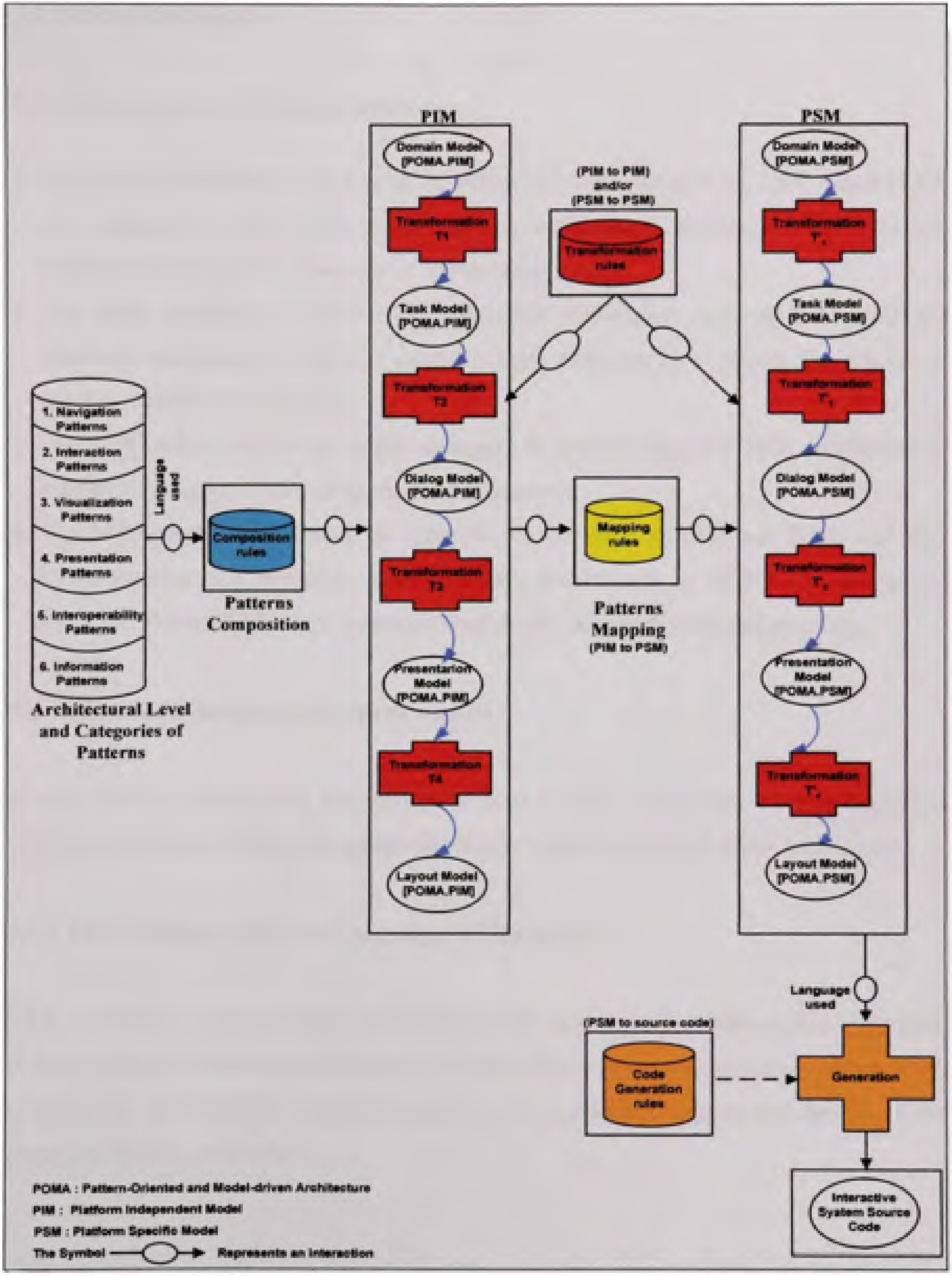

PIM
Domain Model [POMA.PIM]
Transformation T1
Task Model [POMA.PIM]
Transformation T2
Dialog Model [POMA.PIM]
Transformation T3
Presentation Model [POMA.PIM]
Transformation T4
Layout Model [POMA.PIM]
(PIM to PIM) and/or (PSM to PSM)
Transformation rules
PSM
Domain Model [POMA.PSM]
Transformation T'1
Task Model [POMA.PSM]
Transformation T'2
Dialog Model [POMA.PSM]
Transformation T'3
Presentation Model [POMA.PSM]
Transformation T'4
Layout Model [POMA.PSM]
1. Navigation Patterns
2. Interaction Patterns
3. Visualization Patterns
4. Presentation Patterns
5. Interoperability Patterns
6. Information Patterns
Architectural Level and Categories of Patterns
Language used
Composition rules
Patterns Composition
Mapping rules
Patterns Mapping (PIM to PSM)
Language used
(PSM to source code)
Code Generation rules
Generation
Interactive System Source Code
POMA : Pattern-Oriented and Model-driven Architecture
PIM : Platform Independent Model
PSM : Platform Specific Model
The Symbol Represents an interaction

Figure 3.2 POMA architecture for interactive systems development.

3.3 POMA justifications

The justifications for POMA are as follows:

1. The N-tiers architectures such as MVC which is 3-tiers architecture, J2EE which is a 5-tier architecture, and Zachman which is a multi-tiered architecture allow POMA architecture to inherit the concept of architectural levels;
2. The POD architecture based on composition techniques such as behavioral and structural techniques of patterns allows POMA architecture to inherit the concept of these composition techniques;
3. The PSA architecture based on the categories of patterns allows POMA architecture to inherit the concept of this categorization of patterns;
4. The MDA architecture based on types of models such as PIM and PSM, and their transformation and mapping, allows POMA architecture to inherit the concept of models (PIM and PSM) and the concepts of model transformation and mapping.

3.4 POMA specifications and representation

In this research project, two notations have been adopted: eXtensible Markup Language (XML) and Unified Markup Language (UML) for representing the POMA architecture.

3.4.1 The eXtensible Markup Language (XML) notation

XML notation is used to specify and formalize the language for POMA called POMAML (Pattern-Oriented Modeling Architecture Markup Language) (see Appendix III for XML source code of POMAML structural notation) for modeling patterns and models of the proposed POMA architecture.

Indeed, there has been a surge recently in initiatives toward modeling and engineering interactive systems based on model-driven architecture (MDA) using XML.

XML is a meta-language that provides directions for expressing the syntax of Markup languages. Instances of these Markup languages are hierarchically structured documents that typically consist of a content encapsulated within Markup and grammatical instructions on how to process it. The term "document" has a special meaning in XML. A document is a standalone object of representation that acts as a container for processable information. An XML document could, for example, be a physical file on a hard disk or a stream of bytes over a network. Elements and attributes form the most commonly used constructs of an XML document. A given document can conform to the XML Specification in two ways. It can be well formed by allowing further constraints. There are a number of ancillary technologies that strengthen the XML framework. XML Infoset is a description of the information available in a (well-formed) XML document. XML DTD and XML Schema are languages that provide a grammar for structural and data type constraints on the syntax and content of the elements and attributes in XML documents. This allows for verification of formalism or validity in a given document. Namespaces in XML are a mechanism for uniquely identifying elements and attributes of XML documents specific to a markup language, making it possible to create heterogeneous documents that unambiguously mix elements and attributes from multiple different XML documents. Xlink provides the bi-directional linking capabilities necessary for hypertext. XSLT is a style sheet language for transforming XML documents into other formats.

All models are expressed in some notation language. The evolution of notation languages for modeling interactive systems in the last decade has taken place in three orthogonal directions: abstraction; partition (of concerns); standardization. Abstraction has made it possible to define models without getting into the details of implementation or the underlying computing environment. Partition (of concerns) permits describing and dealing with semantically different aspects of the interaction in a changing environment. Standardization has brought some order to the growing complexity of isolated notations that do not always communicate with one another and thereby threaten interoperability among systems. Standardization has also encouraged industry involvement. After embracing a variety of notations and languages over the years for modeling interactive systems, XML has become a popular language in the research community and among practitioners.

3.4.2 The Unified Modeling Language (UML) notation

During the development of interactive systems, the specification of the interactive system and the interaction design are often performed in parallel and therefore must be coordinated. A common notation that can be used and understood by both developers and interface designers would foster integration. This is of particular importance since the interface must eventually be integrated into the rest of the interactive system. UML is a standard language for specifying, visualizing, constructing, and documenting the components of the different types of systems, in particular, interactive systems.

Since its beginnings in 1998, UML has gradually evolved to become an industry standard. UML notation according to (Gamma et al., 1995) and (Muller and Gartner, 2000):

- Is visual modeling which uses the standard graphical notation of patterns;
- Is a communication tool for various patterns;
- Manages the complexity of composed patterns;
- Defines a software architecture;
- Enables and supports reuse;
- Improves the pace at which interactive systems are developed;
- Eases the integration of interfaces with pre-existing modules;
- Decreases interactive system development costs.

UML consists of a set of notations developed to specify and design object-oriented software. UML is made up of a family of notations and models. Among them, class and object diagrams for static domain modeling and use cases and sequence diagrams and activity diagrams, are used for documenting functional requirements. In addition, the system's behavior can be specified using sequence, collaboration, state and activity diagrams.

In summary, this research project uses the UML notation to structure and clarify the proposed POMA architecture.

CHAPTER 4

PATTERNS IN POMA

This chapter presents a detailed description of architectural levels and categories of patterns used in the proposed POMA architecture. The first section presents architectural levels and categories of patterns. The second section describes pattern composition rules (i.e., the relationships between pattern considered in this architecture). The third section describes the pattern mapping rules that enable one to obtain the final models of the proposed architecture.

4.1 Patterns and Pattern-Oriented Architecture

4.1.1 Architectural Levels and categories of patterns

This section presents how the existing categories of patterns can be used as building blocks in the context of the proposed six architectural levels.

This research project has identified at least six architectural levels and six categories of patterns that can be used to create a pattern-oriented interactive system architecture. Table 4.1 illustrates these six levels of POMA architecture for an interactive system, including the corresponding categories of patterns, and gives examples of patterns in each category.

Table 4.1

Architectural levels, categories of patterns and examples

Architectural Level and Category of Patterns	Examples of patterns
Information This category of patterns describes different conceptual models and architectures for organizing the underlying content across multiple pages, servers, and computers. Such patterns provide solutions to questions such as which information can or should be presented on which device. This category of patterns is described in (Jeffrey and Maneesh, 2006).	- Reference Model pattern - Data Column pattern - Cascaded Table pattern - Relational Graph pattern - Proxy Tuple pattern - Expression pattern - Schudler pattern - Operator pattern - Renderer pattern - Production Rule pattern - Camera pattern - Linear Pattern - Hierarchical Pattern - Circular Pattern - Composite Pattern
Interoperability This category of patterns describes decoupling the layers of an interactive system, in particular, between the content, the dialog, and the views or presentation layers. These patterns are generally extensions of the Gamma design patterns, such as MVC (Model, View, and Controller) observer and command action patterns. Communication and interoperability patterns are useful for facilitating the mapping of a design between platforms.	- Adapter pattern - Bridge pattern - Builder pattern - Decorator pattern - Façade pattern - Factory pattern - Method pattern - Mediator pattern - Memento pattern - Prototype pattern - Proxy pattern - Singleton pattern - State pattern - Strategy pattern - Visitor pattern
Visualization This category of patterns describes different visual representations and metaphors for grouping and displaying information in cognitively accessible chunks. They mainly define the format and content of the visualization, i.e. the graphical scene, and as such, relate primarily to data and mapping transforms.	- Favorite Collection pattern - Bookmark pattern - Frequently Visited Page pattern - Navigation Space Map pattern

Table 4.1

Architectural levels, categories of patterns and examples (Continued)

Architectural Level and Category of Patterns	Examples of patterns
Navigation This category of patterns describes proven techniques for navigating within and/or between a set of pages and chunks of information. This list of patterns is far from exhaustive, but helps to communicate the flavor and abstraction level of design patterns for navigation.	- Shortcut pattern - Breadcrumb pattern - Index Browsing pattern - Contextual (temporary) horizontal menu at top pattern - Contextual (temporary) vertical menu at right pattern - Information portal pattern - Permanent horizontal menu at top pattern - Permanent vertical menu at left pattern - Progressive filtering pattern - Shallow menus pattern - Simple universal pattern - Split navigation pattern - Sub-sites pattern - User-driven pattern - Alphabetical index pattern - Key-word search pattern - Intelligent agents pattern - Container navigation pattern - Deeply embedded menus pattern - Hybrid approach pattern - Refreshed shallow vertical menus pattern
Interaction This category of patterns describes the interaction mechanisms that can be used to achieve tasks and the visual effects they have on the scene; as such, they relate primarily to graphical and rendering transforms.	- Search pattern - Executive Summary pattern - Action Button pattern - Guided Tour pattern - Paging pattern - Pull-down Button pattern - Slideshow pattern - Stepping pattern - Wizard pattern
Presentation This category of patterns describes solutions for how the contents or the related services are visually organized into working surfaces, the effective layout of multiple information spaces, and the relationship between them. These patterns define the physical and logical layout suitable for specific interactive systems.	- Carrousel pattern - Table Filter pattern - Detail On Demand pattern - Collector pattern - In place Replacement pattern - List Builder pattern - List Entry View pattern - Overview by Detail pattern - Part Selector pattern - Tabs pattern - Table Sorter pattern - Thumbnail pattern - View

Each of these six categories of patterns is discussed hereunder, and examples are provided.

4.1.1.1 Information patterns

An information pattern, also called an information architectural pattern (Figure 4.1), expresses a fundamental structural organization or schema of information. It provides a set of predefined subsystems (information spaces or chunks), specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them.

An information pattern is everything that happens in a single information space or chunk. With another pattern, the content of a system is organized in a sequence in which all the information spaces or chunks are arranged as peers, and every space or chunk is accessible by all the others. This is very common on simple sites where there are only a few standard topics, such as: Home, About Us, Contact Us, and Products. Information which naturally flows as a narrative, a time line, or in a logical order is ideal for sequential treatment. An index structure is like the flat structure, with an additional list of contents. An index is often organized in such a way as to make its content easier to find. For example, a list of files in a Web directory (the index page), an index of people's names ordered by last name. Dictionaries and phone books are both very large indices.

The Hub-and-Spoke pattern is useful for multiple distinct linear workflows. A good example would be an email system where the user returns to his inbox from several points, e.g. after reading a message, after sending a message, or after adding a new contact. A multi-dimensional hierarchy is one in which there are many ways to browse the same content. In a way, several hierarchies may coexist, overlaid on the same content. The structure of the content can appear to be different, depending on the user's task (search, browse). A typical example would be a site like Amazon, which lets one browse books by genre or by title, and also allows search by keyword. Each of these hierarchies corresponds to a property of the content, and each can be useful, depending on the user's situation. A strict hierarchy is a specialization of a multi-dimensional hierarchy, and describes a system where a lower-level page can only be accessed via its parent.

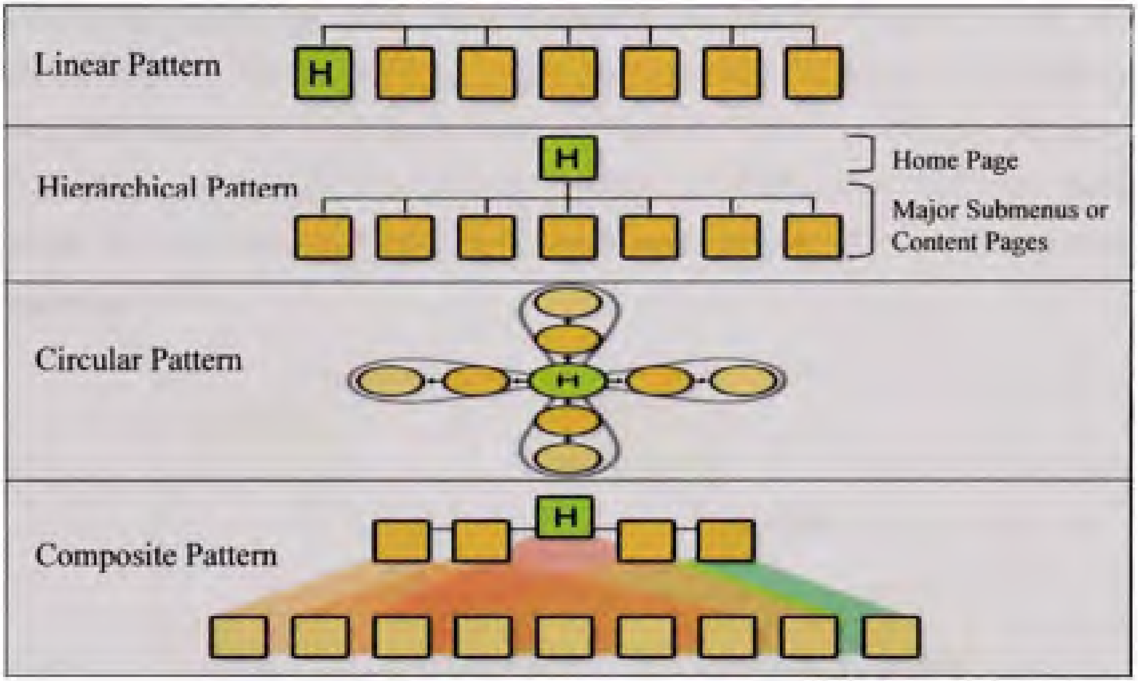

Linear Pattern
H
Hierarchical Pattern
H
Home Page
Major Submenus or Content Pages
Circular Pattern
H
Composite Pattern
H

Figure 4.1 Examples of Information Patterns.

4.1.1.2 Interoperability patterns

Interoperability patterns are useful for decoupling the organization of these different categories of patterns, for the way information is presented to the user, and for the user who interacts with the information content. Patterns in this category generally describe the capability of different programs to exchange data, via a common set of exchange formats, to read and write under the same file formats, and to use the same protocols.

Gamma *et al.* (1995) offer a large catalog of patterns for dealing with such problems. Examples of patterns applicable to interactive systems include: Adapter, Bridge, Builder, Decorator, Factory Method, Mediator, Memento, Prototype, Proxy, Singleton, State, Strategy, and Visitor (Gamma et al., 1995).

The Adapter pattern is very common, not only to remote client/server programming, but to any situation in which there is one class and it is desirable to reuse that class, but where the system interface does not match the class interface. Figure 4.2 illustrates how an adapter works. In this figure, the Client wants to invoke the method *Request()* in the Target interface. Since the Adaptee class has no *Request()* method, it is the job of the Adapter to convert the request to an available matching method. Here, the Adapter converts the method *Request()* call into the Adaptee method *specificRequest()* call. The Adapter performs this conversion for each method that needs adapting. This is also known as *Wrappering*.

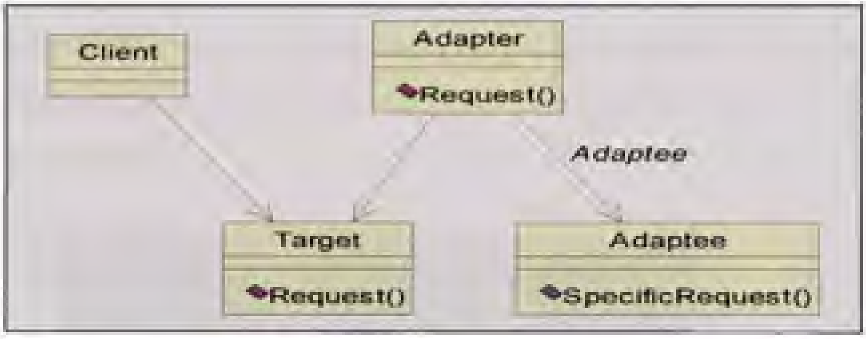


Figure 4.2 Adapter pattern.

4.1.1.3 Visualization patterns

Information visualization patterns allow users to browse information spaces and focus quickly on items of interest. Visualization patterns can help to avoid an information overload, a fundamental issue to tackle, especially for large databases, Web sites, and portals, as they can access millions of documents. The designer must consider how best to map the contents into a visual representation which conveys information to the user while facilitating exploration of the content. In addition, the designer must undertake dynamic actions to limit the amount of information the user receives, while at the same time keeping the user informed about the content as a whole. Several information visualization patterns generally combine in such a way that the underlying content can be organized into distinct conceptual spaces or working surfaces which are semantically linked to one another.

For example, depending on the purpose of the site, users can access several kinds of "pages", such as articles, URLs and products. They typically collect several of these items for a specific task, such as comparing, buying, going to a page, sending a page to others. Users must be able to visualize their "collection".

The following are some of the information visualization patterns for displaying such collections: Favorite, Bookmark, Frequently Visited Page, Preferences, and Navigable Spaces Map. This category of patterns provides a map to a large amount of content which can be too large to be presented reasonably in a single view. The content can be organized into distinct conceptual spaces or working surfaces which are semantically linked, so that it is natural and meaningful to go from one to another. The map in Figure 4.3 is an example of this category of patterns.

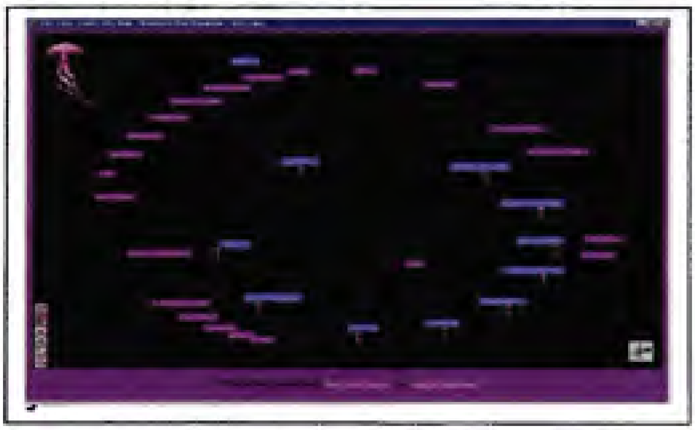


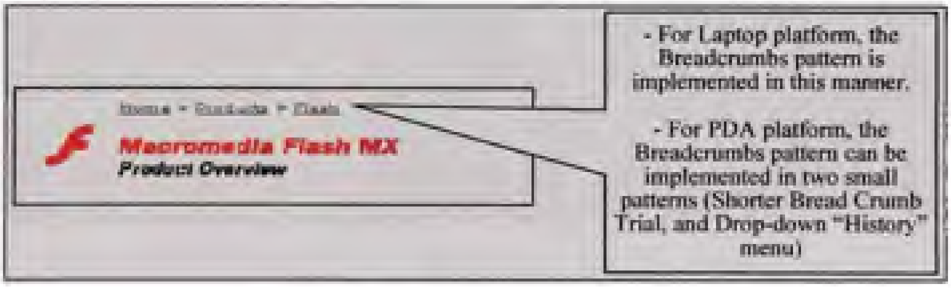
Figure 4.3 The Navigation Spaces Map pattern implemented using Tree Hyperbolic, a sophisticated visualization technique.

4.1.1.4 Navigation patterns

Navigation patterns help the user move easily and in a straightforward manner between information chunks and their representations. They can obviously reduce the user's memory load (Nielsen, 1999) and (Lynch and Horton, 1999). See (Tidwell, 1997), (Welie, 1999), (Engelberg and Seffah, 2002) and (Garrido et al., 1997) for an exhaustive list of navigation patterns.

The Linear Navigation pattern is suitable when a user wants a simple way to navigate from one page to the next in a linear fashion, i.e. move through a sequence of pages.

The Index Browsing pattern is similar to the Linear Navigation pattern and allows a user to navigate directly from one item to the next and back. The ordering can be based on a ranking. For every item presented to the user, a navigation widget allows the user to choose the next or previous item in the list. The ordering criterion should be visible (and be user-configurable). To support orientation, the current item number and total number of items should be clearly visible. A breadcrumb (Figure 4.4) is a widely used pattern which helps users to know where they are in a hierarchical structure and to navigate back up to higher levels in the hierarchy. It shows the hierarchical path from the top level to the current page and makes each step clickable.



- For Laptop platform, the Breadcrumbs pattern is implemented in this manner.

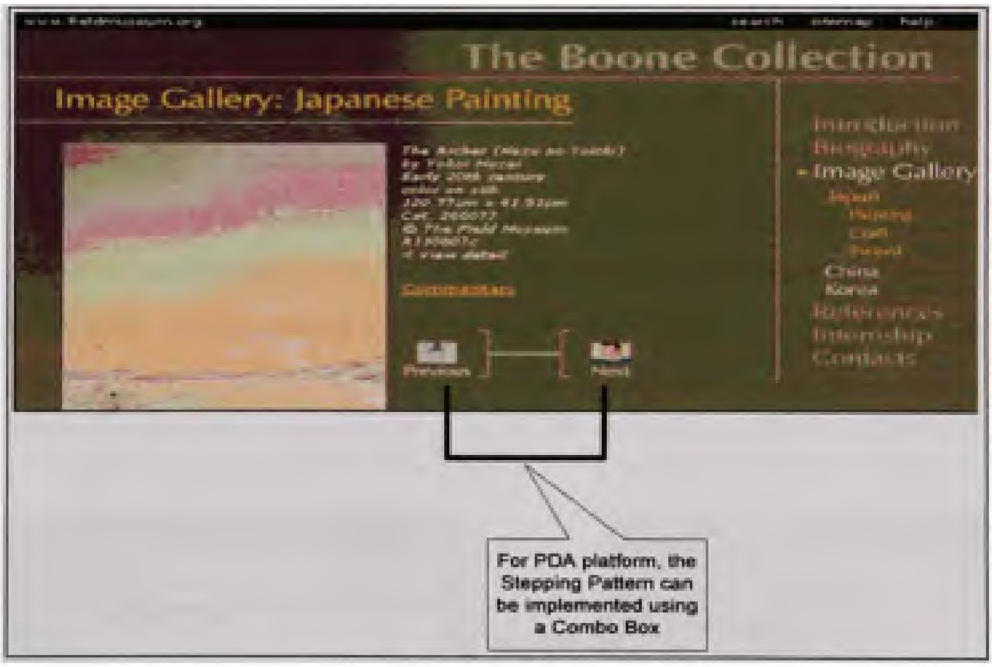
- For PDA platform, the Breadcrumbs pattern can be implemented in two small patterns (Shorter Bread Crumb Trial, and Drop-down "History" menu)

Figure 4.4 Breadcrumb Pattern.
(Extracted from Swish Zone Website)

4.1.1.5 Interaction patterns

This category of interaction patterns provides basic information on interaction style, mainly on how to use controls such as buttons, lists of items, menus and dialog boxes. This category of patterns is employed whenever users need to take an important action that is relevant in the current context of the page being viewed. Users must be made aware of the importance of the action in relation to other actions on the page or site.

To view/act on a linear-ordered set of items, the Stepping pattern (Figure 4.5) allows users to go to the next and previous task or object by clicking on the 'Next' or 'Previous' links. The 'next' link takes the users to the next item in the sequence, while the 'previous' link takes them a step back. It is recommended that a 'next' or 'previous' link be placed close to the object to which it belongs, preferably above the object so that users do not have to scroll to it. One must sure the next/previous links are always placed in the same location, so that users clicking through a list do not have to move the mouse pointer. The convention, at least in western cultures, is to place the 'Previous' link on the left and the 'Next' link on the right.



For PDA platform, the Stepping Pattern can be implemented using a Combo Box

Figure 4.5 Stepping pattern.
(Extracted from Field Museum Website)

4.1.1.6 Presentation patterns

The authors of technical documents discovered long before interactive systems were invented that users appreciate short "chunks" of information (Horton, 1994). Patterns in this category, called Presentation patterns, also suggest different ways for displaying chunks of information and ways for grouping them in pages. Presentation patterns also define the look and feel of interactive systems, while at the same time defining the physical and logical layout suitable for specific systems, such as home pages, lists, and tables. For example, how long does it take to determine whether or not a document contains relevant information? This question is a critical design issue, in particular for resource-constrained (small) devices.

Patterns in this category use a grid, which is a technique taken from print design, but which is easily applicable to interactive system design as well. In its strictest form, a grid is literally a grid of X by Y pixels. The elements on the page are then placed on the cell borderlines and aligned overall on horizontal and vertical lines. A grid is a consistent system in which to place objects. In the literature on print design, there are many variations of grids, most of them based on modular and column grids. Often, a mix of both types of grids will be used. An example of a grid in Figure 4.6 is used to create several dialog box patterns.

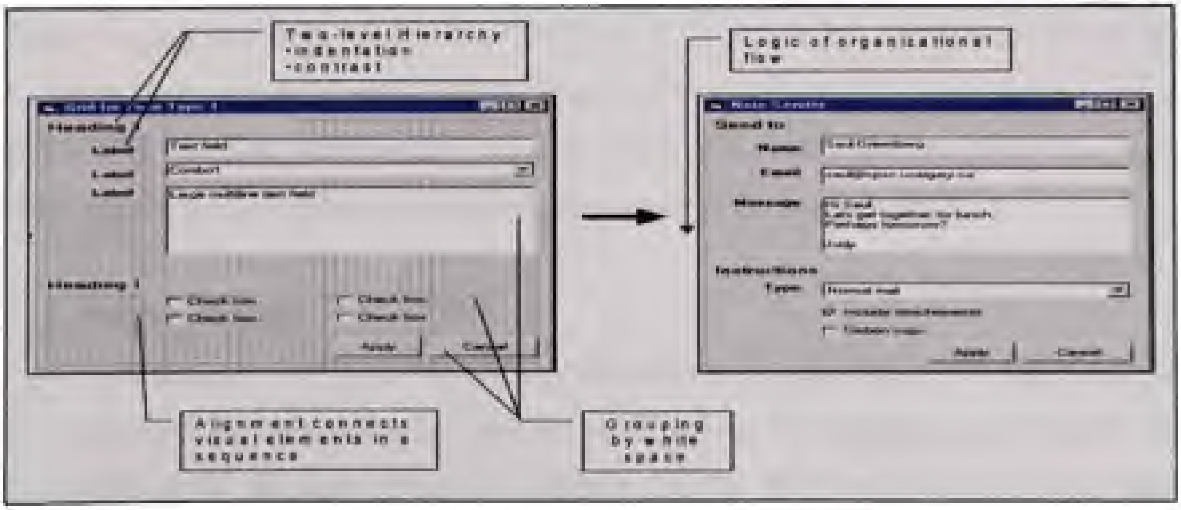
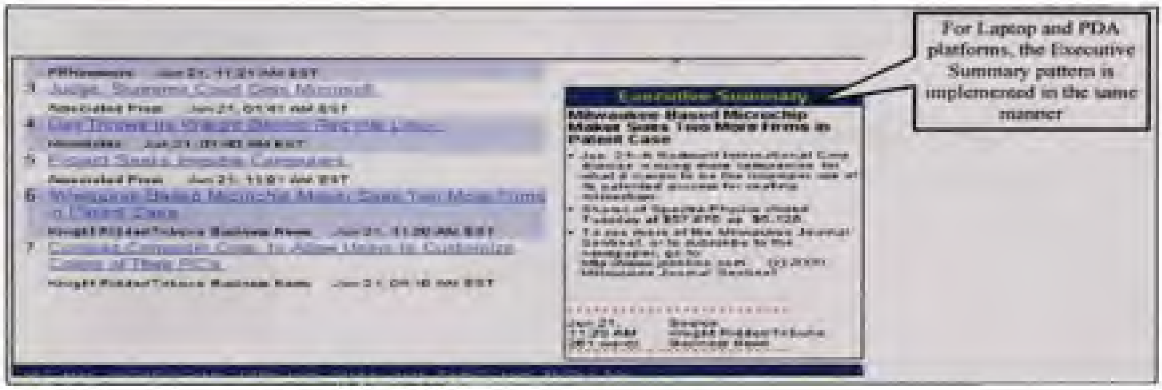

Two-level Hierarchy
-indentation
-contrast
Logic of organizational flow
Heading
Label
Label
Label
Heading
Check box
Check box
Check box
Check box
Apply
Cancel
Send to
Name
Email
Message
Instructions
Type
Apply
Cancel
Alignment connects visual elements in a sequence
Grouping by white space

Figure 4.6 An example of a grid.

An example of these types of patterns is the Executive Summary pattern. The Executive Summary pattern gives users a preview of the underlying information before they spend time downloading, browsing, and reading large amounts of information (Figure 4.7).



For Laptop and PDA platforms, the Executive Summary pattern is implemented in the same manner

Figure 4.7 Example of structural patterns: Executive Summary Pattern.
(Extracted from CBC Website)

4.1.2 Pattern Composition

A platform-independent pattern-oriented design exploits several relationships between patterns. Gamma *et al.* (1995) emphasize that defining the list of related patterns as part of the description of a pattern is a key notion in the composition of patterns and their uses. Zimmer (1994) implements this idea by dividing the relations between the patterns of the Gamma catalog into three types: "X is similar to Y", "X uses Y", and "Variants of X use Y". These types are, in practice, relationships between patterns in a specific context; in other words, they are relationships between instances of patterns. Based on Zimmer's (1994) definitions, five types of relationships between patterns are defined.

1. **Similar Pattern**

   Two patterns (X, Y) are similar, or equivalent, if, and only if, X and Y can be replaced by each other in a certain composition. This means that X and Y are patterns of the same category and they provide different solutions to the same problem in the same context. As illustrated in Figure 4.8, the *Index Browsing* and

Menu Bar patterns are similar. They both provide navigational support in the context of a medium-sized interactive system.

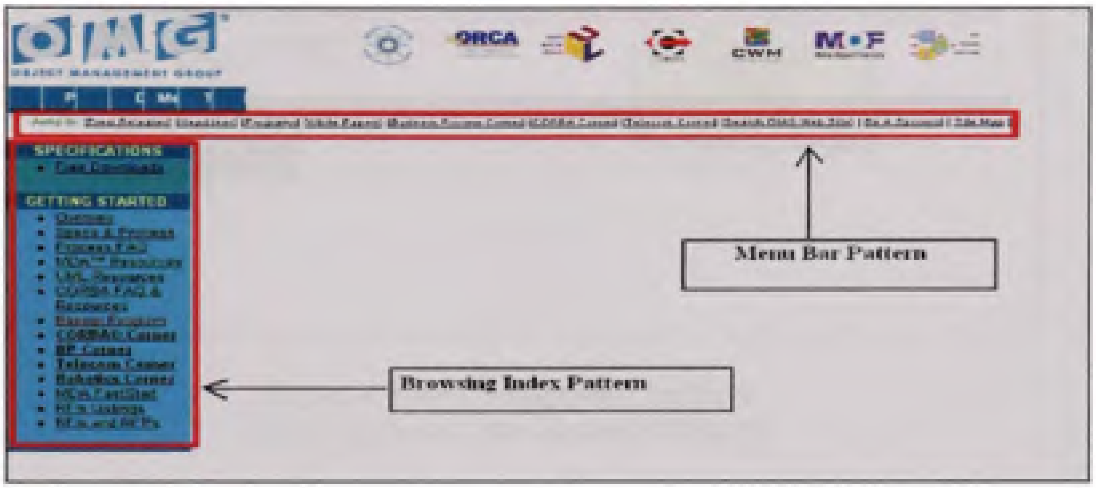

OMG
OBJECT MANAGEMENT GROUP
ORCA
CWH
M•F
SPECIFICATIONS
GETTING STARTED
Menu Bar Pattern
Browsing Index Pattern

Figure 4.8 Similar Pattern.
(Extracted from OMG Website)

2. **Competitor Pattern**

   Two patterns (X, Y) are competitors if X and Y cannot be used at the same time for designing the same artifact relationship that applies to two patterns of the same pattern category. Two patterns are competitors if, and only if, they are similar and interchangeable. For example, the Web patterns *Convenient Toolbar* and *Index Browsing* are competitors (Figure 4.9). The *Index Browsing* pattern can be used as a shortcut toolbar that allows a user to directly access a set of common services from any interactive system. The *Convenient Toolbar*, which provides the same solution, is generally considered more appropriate.

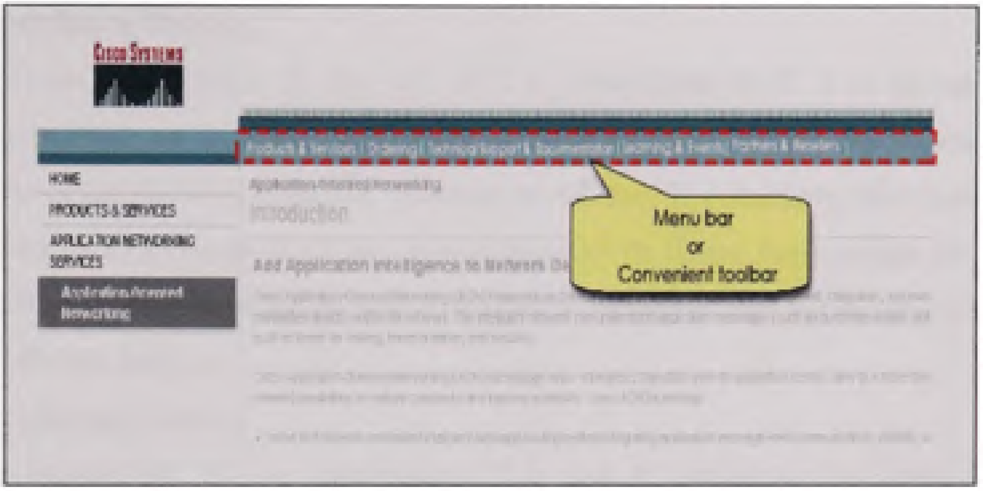

Cisco Systems
Products & Services | Ordering | Technical Support & Documentation | Learning & Events | Partners & Resellers |
HOME
PRODUCTS & SERVICES
APPLICATION NETWORKING SERVICES
Application-Oriented Networking
Application-Oriented Networking
Introduction
Add Application Intelligence to Network De
Menu bar
or
Convenient toolbar

Figure 4.9 Two Competitor Pattern.
(Extracted from Cisco Systems Website)

3. **Super-ordinate Pattern**

   A pattern X is a super-ordinate of pattern Y, which means that pattern Y is used as a building block to create pattern X. An example is the *Home Page* pattern, which is generally composed of several other patterns (Figure 4.10).

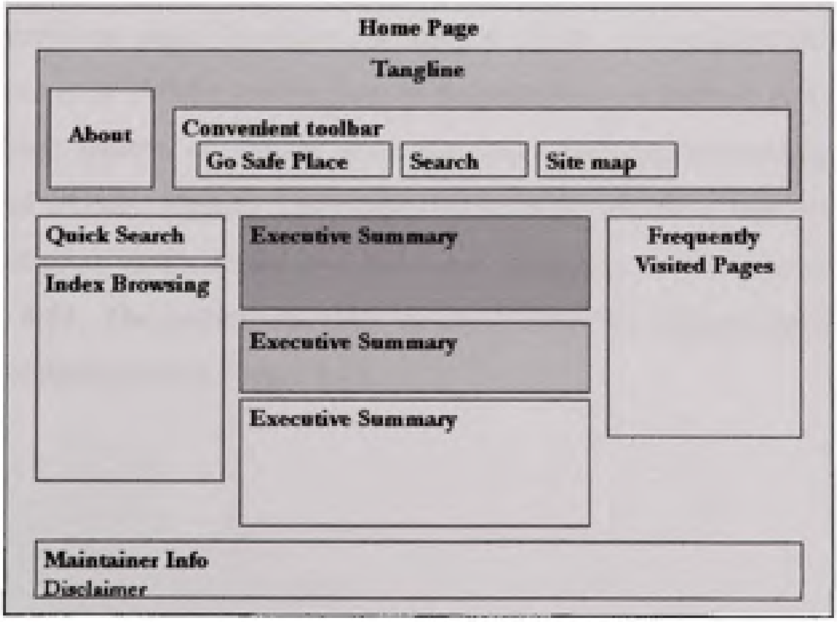

Home Page
Tangline
About
Convenient toolbar
Go Safe Place
Search
Site map
Quick Search
Index Browsing
Executive Summary
Executive Summary
Executive Summary
Frequently Visited Pages
Maintainer Info
Disclaimer

Figure 4.10 A Home Page Design Pattern using others patterns.

4. **Sub-ordinate Pattern**

   (X, Y) are sub-ordinate if, and only if, X is embeddable in Y. Y is also called a super-ordinate of X. This relationship is important in the process of mapping pattern-oriented design from one platform to another. For example, the *Convenient Toolbar* pattern (Figure 4.10) is a sub-ordinate of the *Home Page* pattern for either a PDA or desktop interactive system. Implementations of this pattern are different for different devices.

5. **Neighboring Pattern**

   Two patterns (X, Y) are neighboring if X and Y belong to the same pattern category. For example, the Sequential and Hierarchical patterns are neighboring because they belong to the same category of patterns, and neighboring patterns may include the set of patterns for designing a specific page such as a home page (Figure 4.10).

This research project investigates how these categories of proven design patterns are "composed" and "mapped" into reliable, robust large-scale interactive systems.

Figure 4.10 in previous page illustrates the details of the six architectural levels and categories of patterns of POMA architecture in relationships as defined and described at section 4.1.1. Each pattern of Figure 4.10 can have two representations: Structural representation and POMA Markup Language (POMAML) structure representation. The structural representation is illustrated by UML class diagrams in the Figure 4.11, Figure 4.12 and Figure 4.13. The pattern structure of the POMAML schema representation is illustrated by a notation given in Figure 4.13.

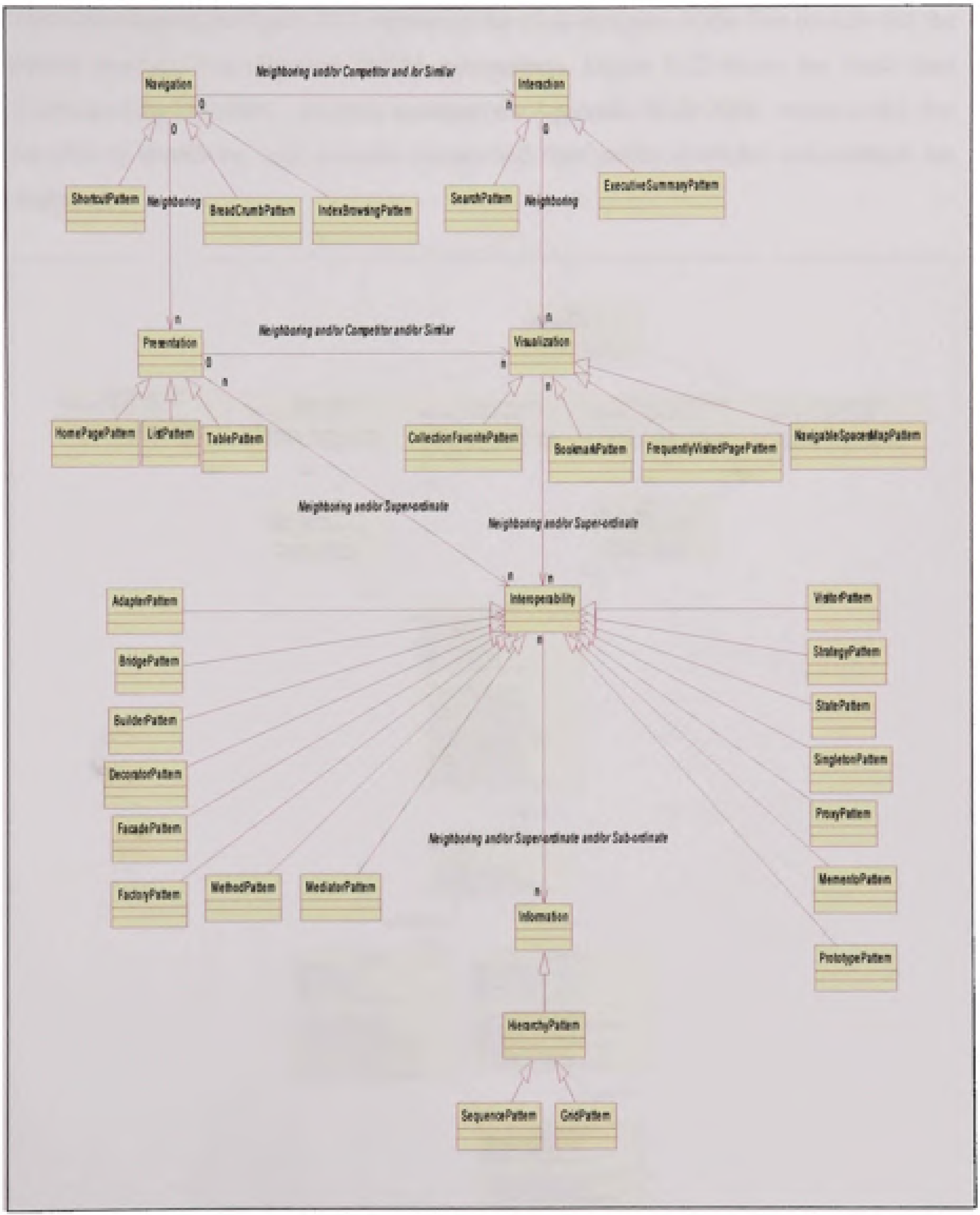

Navigation
Neighboring and/or Competitor and /or Similar
Interaction
0
n
0
0
ShortcutPattern
Neighboring
BreadCrumbPattern
IndexBrowsingPattern
SearchPattern
Neighboring
ExecutiveSummaryPattern
n
n
Presentation
Neighboring and/or Competitor and/or Similar
Visualization
0
n
n
n
HomePagePattern
ListPattern
TablePattern
CollectionFavoritePattern
BookmarkPattern
FrequentlyVisitedPagePattern
NavigableSpacesMapPattern
Neighboring and/or Super-ordinate
Neighboring and/or Super-ordinate
n
n
AdapterPattern
Interoperability
VisitorPattern
n
BridgePattern
StrategyPattern
BuilderPattern
StatePattern
DecoratorPattern
SingletonPattern
FacadePattern
ProxyPattern
Neighboring and/or Super-ordinate and/or Sub-ordinate
FactoryPattern
MethodPattern
MediatorPattern
MementoPattern
n
Information
PrototypePattern
HierarchyPattern
SequencePattern
GridPattern

Figure 4.11 UML Class Diagram of architectural level and categories of Patterns of POMA for Interactive System.

The class diagram in Figure 4.12 represents the class structure of the five models and the pattern structure that represent POMA components. Figure 4.12 shows the basic class structure of the POMAML structure notation (see Appendix III for XML source code). For the sake of simplicity, only concrete classes and their public attributes and methods are displayed.

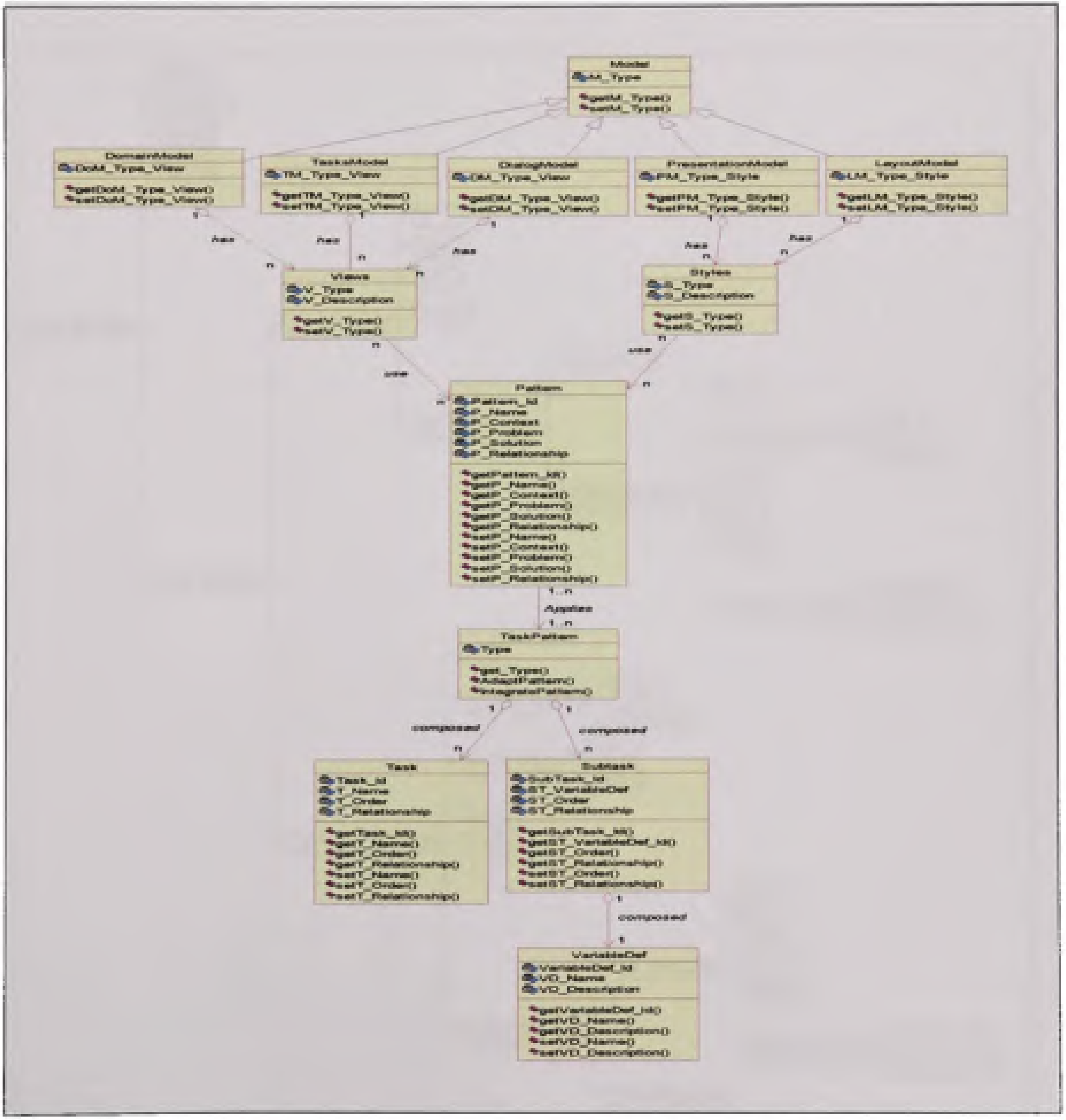

Model
M_Type
getM_Type()
setM_Type()
DomainModel
DoM_Type_View
getDoM_Type_View()
setDoM_Type_View()
TasksModel
TM_Type_View
getTM_Type_View()
setTM_Type_View()
DialogModel
DM_Type_View
getDM_Type_View()
setDM_Type_View()
PresentationModel
PM_Type_Style
getPM_Type_Style()
setPM_Type_Style()
LayoutModel
LM_Type_Style
getLM_Type_Style()
setLM_Type_Style()
has
has
has
has
has
1
n
Views
V_Type
V_Description
getV_Type()
setV_Type()
Styles
S_Type
S_Description
getS_Type()
setS_Type()
use
use
Pattern
Pattern_Id
P_Name
P_Context
P_Problem
P_Solution
P_Relationship
getPattern_Id()
getP_Name()
getP_Context()
getP_Problem()
getP_Solution()
getP_Relationship()
setP_Name()
setP_Context()
setP_Problem()
setP_Solution()
setP_Relationship()
1..n
Applies
1..n
TaskPattern
Type
get_Type()
AdaptPattern()
IntegratePattern()
composed
composed
Task
Task_Id
T_Name
T_Order
T_Relationship
getTask_Id()
getT_Name()
getT_Order()
getT_Relationship()
setT_Name()
setT_Order()
setT_Relationship()
SubTask
SubTask_Id
ST_VariableDef
ST_Order
ST_Relationship
getSubTask_Id()
getST_VariableDef_Id()
getST_Order()
getST_Relationship()
setST_Order()
setST_Relationship()
composed
VariableDef
VariableDef_Id
VD_Name
VD_Description
getVariableDef_Id()
getVD_Name()
getVD_Description()
setVD_Name()
setVD_Description()

Figure 4.12 Class structure of POMA's Models and Patterns.

POMAML is an acronym for Pattern-Oriented and Model-driven Architecture Markup Language and is graphically XML structure displayed in Figure 4.13 described in the Figure 4.11 and Figure 4.12 . In other words, Figure 4.13 is a form or structure of XML notation that is used to represent patterns used in Figure 4.11 and Figure 4.12. POMAML XML notation for tasks and feature patterns were developed (see Appendix III for the XML source code of POMAML structural notation).

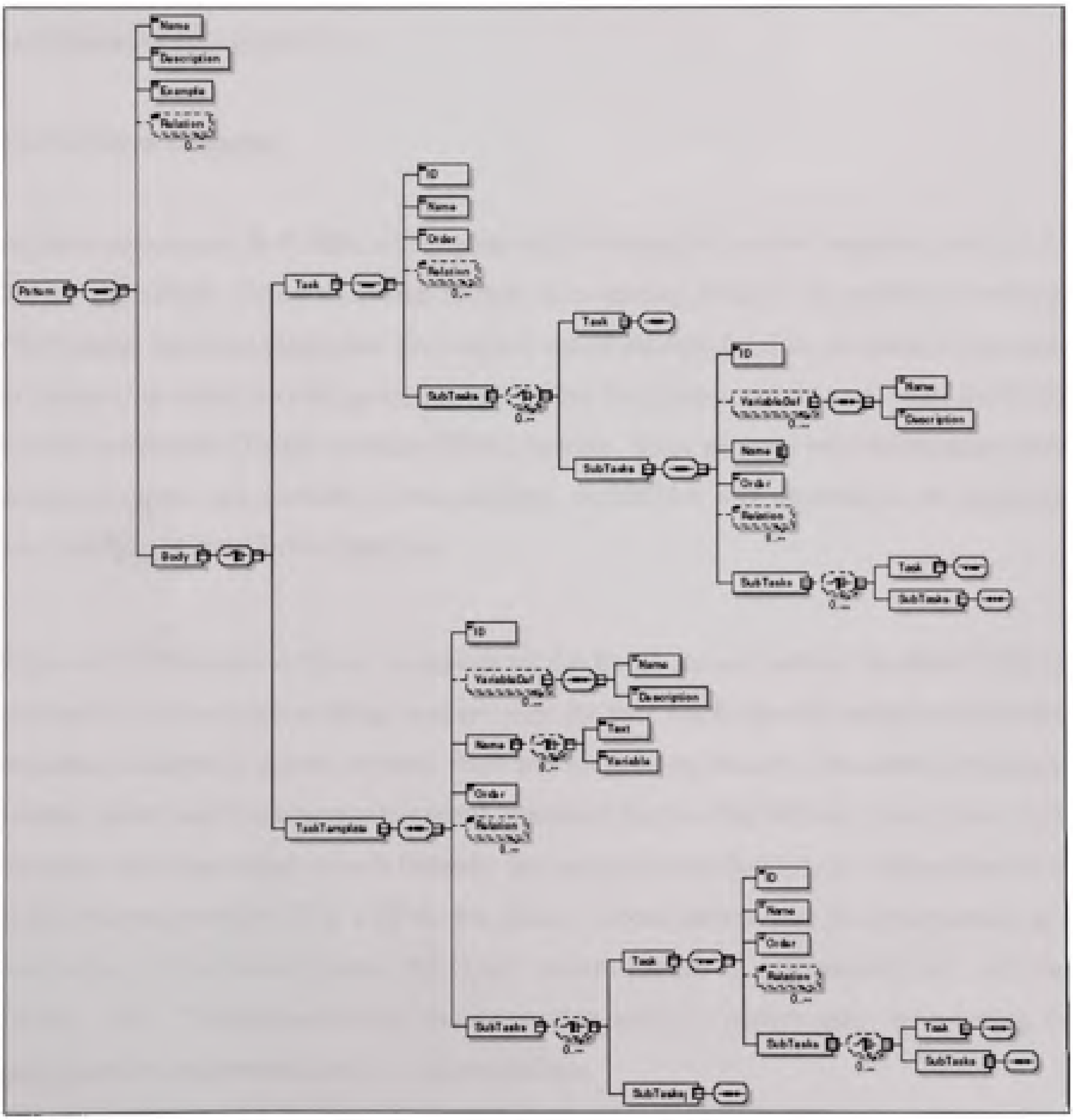

Pattern
Name
Description
Example
Relation
0..∞
Body
Task
ID
Name
Order
Relation
0..∞
SubTasks
0..∞
Task
SubTasks
ID
VariableDef
0..∞
Name
Description
Name
Order
Relation
0..∞
SubTasks
0..∞
Task
SubTasks
TaskTemplate
ID
VariableDef
0..∞
Name
Description
Name
0..∞
Text
Variable
Order
Relation
0..∞
SubTasks
0..∞
Task
ID
Name
Order
Relation
0..∞
SubTasks
0..∞
Task
SubTasks
SubTasks

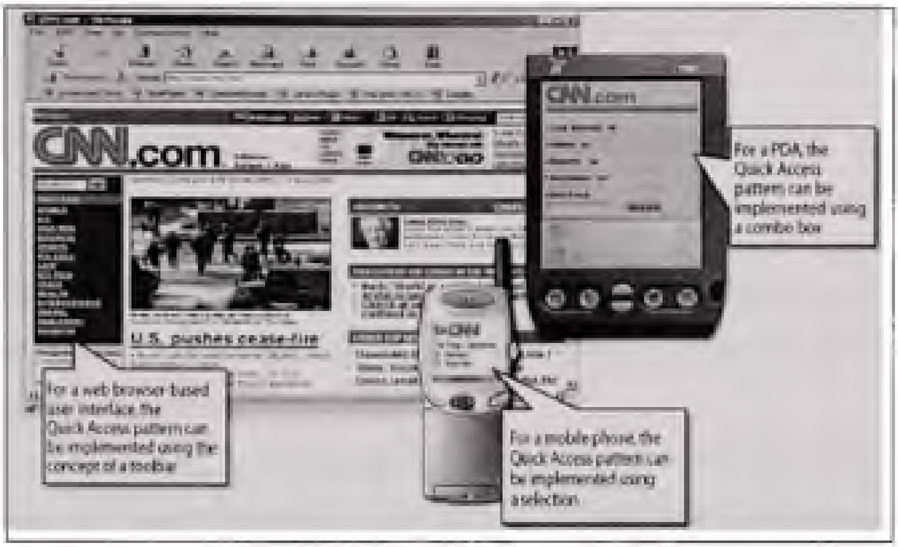
Figure 4.13 Pattern structure of the POMAML Markup Language.

The POMAML Schema (Figure 4.13) consists of the classic elements of patterns like Name, Problem, Context, Solution and Rational. However, these attributes are primarily used only to select an appropriate pattern. The implementation of the pattern has been formalized in the "Body". At this point, one should distinguish between Task and TaskTemplates. Tasks are further decomposed into SubTasks and contain no variable parts. Thus they can be adopted 1:1 without further adaptation. On the contrary, TaskTemplates are hierarchically structured as well, but also contain variable definitions and variables and must therefore be adapted first.

4.1.3 Pattern mapping

Another component in POMA architecture is the concept of *pattern mapping* (section 3.1 for the definition). Using a desktop system as a starting point, it is possible to redesign PSM model for other platforms. The original set of patterns used in the system is mapped or replaced in order to redesign and re-implement the system and, in particular, the UI for mobile or Personal Digital Assistant (PDA) systems. Since patterns hold information about design solutions and context of use, platform capabilities and constraints are implicitly addressed in the transformed patterns.

Figure 4.14 illustrates different mappings of the *Quick Access pattern* for three different platforms. This navigation design pattern helps the user reach specific pages, which reflect important interactive system content, from any location on the site. For a news interactive system, direct and quick access to central interfaces such as *Top Stories*, *News*, *Sports*, and *Business* can be provided. A web browser, for example, on a desktop, is implemented as an *index browsing toolbar*. For a PDA, the Quick Access pattern can be implemented as a *combo box*. For a mobile phone, the Quick Access pattern is implemented as a *selection* (Welie, 1999). Pattern descriptions should provide advice to pattern users for selecting the most suitable implementation for a given platform.



For a web browser-based user interface the Quick Access pattern can be implemented using the concept of a toolbar

For a PDA, the Quick Access pattern can be implemented using a combo box

For a mobile phone, the Quick Access pattern can be implemented using a selection

Figure 4.14 The Web Convenient Toolbar pattern implementations and Look and Feels for different platforms.
(Extracted from the CNN Website)

To illustrate pattern mapping, a description is given here of the effect of screen size on selection and use of patterns. Different platforms use different screen sizes, and these different screen sizes afford different types and variants of patterns. The problem to resolve when mapping a pattern-oriented design (POD) is how the change in screen size between two platforms affects redesign at the pattern level. The amount of information that can be displayed on a given platform screen is determined by a combination of area and the number of pixels. The total difference in information capacity between platforms will be somewhere between these two measures: 20 times the area and 10 times the pixels.

To map the desktop display architecture to the PDA display architecture, the options are as follows:

1. Reduce the size of the architecture; it is necessary to reduce significantly both the number of pages and the quantity of information per page;
2. Hold the architecture size constant (i.e. topics or pages); it is necessary to significantly reduce the quantity of information per page (by a factor of about 10 to 20);
3. Retain all the information in the desktop architecture; it is necessary to significantly

increase the size of the architecture, since the PDA can hold less information per page.

The mapping choice will depend on the size of the architecture and the value of the information:

- For small desktop architectures, the design strategy can be weighted either toward reducing information, if the information is not important, or toward increasing the number of pages if the information is important;
- For medium and large desktop architectures, it is necessary to weight the design strategy heavily toward reducing the quantity of information, otherwise the architecture size and number of levels would rapidly explode out of control.

Finally, one can consider mapping patterns and graphical objects in the context of the amount of change that must be applied to the desktop design or architecture to fit it into a PDA format. The following is the list of suggested mapping rules:

1. **Identical:** No change to the original design. For example, drop-down menus can usually be copied from a desktop to a PDA without any design changes;
2. **Scalable:** Changes to the size of the original design or to the number of items in the original design. For example, a long horizontal menu can be adapted to a PDA by reducing the number of menu elements;
3. **Multiple:** Repeating the original design, either simultaneously or sequentially. For example, a single long menu can be transformed into a series of shorter menus;
4. **Fundamental:** Change the nature of the original design. For example, permanent left-hand vertical menus are useful on desktop displays, but are not practical on most PDAs. In mapping to a PDA, left-hand menus normally need to be replaced with an alternative such as a drop-down menu.

These mapping rules can be used by designers in the selection of patterns, especially when different patterns apply for one platform but not for another, when the cost of adapting or purchasing a pattern is high, or when the applicability of a pattern (knowing how and when to apply a pattern) is questionable.

This list of four mapping rules is especially relevant to the automation of cross-platform design mapping, since the designs that are easiest to map are those that require the least mapping. The category of patterns therefore identifies where human intervention will be needed for design decisions in the mapping process. In addition, when building a desktop design for which a PDA version is also planned, the category of patterns indicates which patterns to use in the desktop design to allow easy mapping to the PDA design.

Figure 4.15 illustrates some of the navigation design patterns used in the home page of a desktop-based system. Once these patterns are identified in the desktop-based system, they can be mapped or replaced by others in a PDA version.

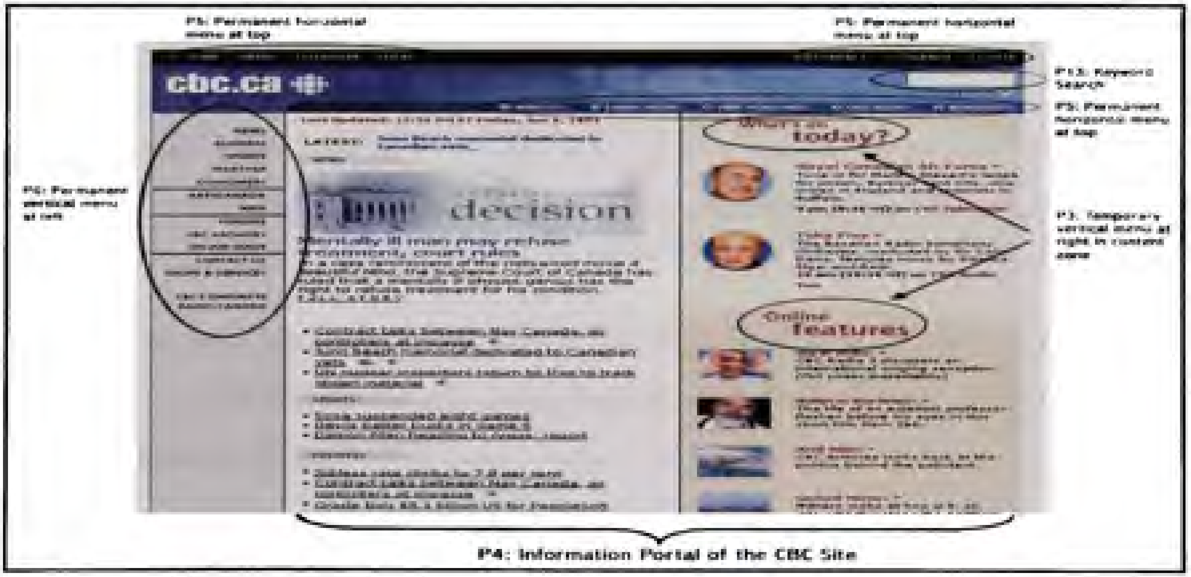

P5: Permanent horizontal menu at top
P5: Permanent horizontal menu at top
cbc.ca
P13: Keyword Search
P5: Permanent horizontal menu at top
P6: Permanent vertical menu at left
decision
today?
P3: Temporary vertical menu at right in content zone
features
P4: Information Portal of the CBC Site

Figure 4.15 Examples of patterns.
(Extracted from the CBC News Website)

Figure 4.16 demonstrates the redesigned interface of the CBC site for migrating to a PDA platform. The permanent horizontal menu pattern at the top (P5) in the original desktop UI were repositioned to a shorter horizontal menu pattern (P5s). In order to accommodate this change on the small PDA screen, the three different horizontal menus had to be shortened, and only important navigation items were used. The keyword search pattern (P13) remains as a keyword search. The permanent vertical menu on the left (P6) was redesigned to a drop-down menu (P15). The drop-down menu in the PDA design also includes the menu headings, "***What's on today?***" and "***Online features***" from the temporary vertical menu (P3) in the original desktop design. Finally, the information portal (P4), which is the first item that captures the user's attention, was redesigned as a smaller information portal (P4s).

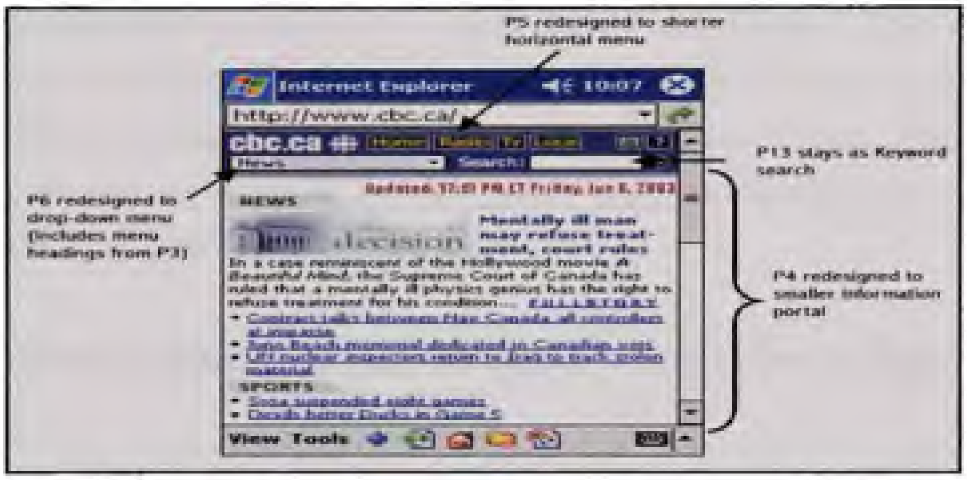

P5 redesigned to shorter horizontal menu
Internet Explorer
10:07
http://www.cbc.ca/
cbc.ca
P13 stays as Keyword search
P6 redesigned to drop-down menu (includes menu headings from P3)
NEWS
SPORTS
P4 redesigned to smaller information portal
View Tools

Figure 4.16 Migration of the CBC site to a PDA Platform using Pattern Mapping.
(Extracted from the CBC News Website)

What has just been illustrated in this section and the examples in Figure 4.14, Figure 4.15 and Figure 4.16 can be characterized in the form of composed and mapped pattern-oriented design architecture (Figure 4.17).

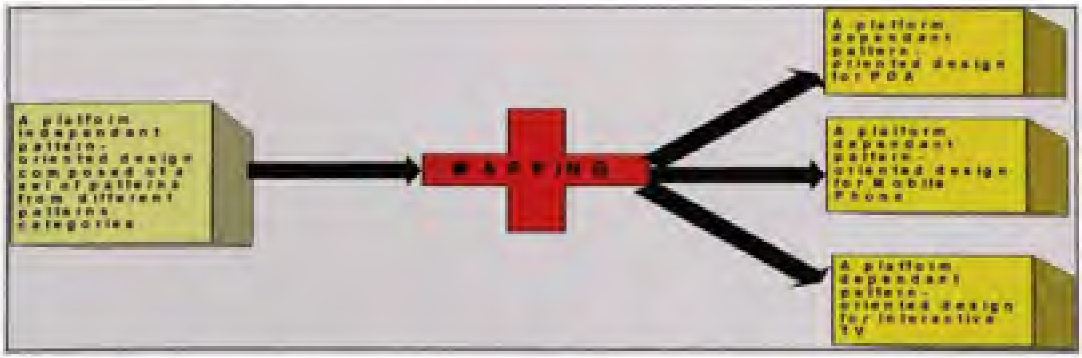

A platform independent pattern-oriented design composed of a set of patterns from different patterns categories
MAPPING
A platform dependant pattern-oriented design for PDA
A platform dependant pattern-oriented design for Mobile Phone
A platform dependant pattern-oriented design for Interactive TV

Figure 4.17 Pattern-Oriented Composition and Mapping Design Architecture.

4.2 Summary of chapter

This chapter has focused on an architectural level and categories of patterns that combine one key approach: pattern-oriented. Architectural levels and categories of patterns have been described (Navigation patterns, Interaction patterns, Visualization patterns, Presentation patterns, Interoperability patterns, and Information patterns) as well as the different relationships between patterns. Their relationships are used to combine using the composition rules described in section 4.1.2 such as similar, competitor, super-ordinate, sub-ordinate and neighbouring to create a platform independent model (PIM) and to map several types of patterns to create a platform specific model (PSM) design using mapping rules described in section 4.1.3 such as identical, scalable, multiple and fundamental for interactive systems, as well as to generate specific implementations suitable to different platforms from the same pattern-oriented design.

In chapter 5 which follows, model categories used in POMA architecture are presented.

CHAPTER 5

MODELS IN POMA

This chapter presents a detailed description of five levels and categories of models used by POMA. The first section describes model categorization. The second section defines model transformation rules which apply for each type of model, [POMA.PIM] or [POMA.PSM]. These rules enable one to build a relationship between models of each category, i.e., models [POMA.PIM] and [POMA.PSM]. The last section explains the scope of the source code generation phase in POMA.

5.1 Model Categorizations

A categorization of models is proposed here. Examples of models are also presented to illustrate the need to map and/or to transform several types of models to provide solutions to problems on the six architectural levels. This section describes how these models can be used at six levels of the proposed POMA architecture to create a model-driven architecture for interactive systems.

The focus is on a subset of the proposed models by this research project and consists of:

- A domain model;
- A task model;
- A dialog model;
- A presentation model;
- A layout model.

5.1.1 Domain model

The Domain model is sometimes called a business model. It encapsulates the important entities of a system domain along with their attributes, methods, and relationships (Schlungbaum, 1996) and (Sinnig, 2004). Within the scope of user interface (UI) development, it defines the objects and functionalities accessed by the user via the interface. Such a model is generally developed using the information collected during the business and functional requirements stage. The information defines the list of data and features or operations to be performed in various ways, i.e. by different users on different platforms.

The first Model-based approaches use a Domain model to drive the UI at runtime. In this context, the Domain model would describe the interactive system in general, and include some specific information for the UI. For example, the Domain model (Schlungbaum, 1996) would include:

- A class hierarchy of objects which exist in the interactive system;
- Properties of the objects;
- Actions which can be performed on the objects;
- Units of information (parameters) required by the actions;
- Pre- and post-conditions for the actions.

Consequently, the only real way to integrate UI and system development is the simultaneous use of the data model. This is why recent model-based approaches include a Domain model known from the system engineering methods. Four other models: Task, Dialog, Presentation, and Layout, have the Domain model as an input.

5.1.2 Task model

The Task model makes it possible to describe how tasks can be performed to reach the user's goals when using an interactive system (Paternò, 2000). Using this model, designers can develop integrated descriptions of the system from a functional and interactive point of view. Task models are typically tasks and subtasks hierarchically decomposed into atomic actions (Souchon et al., 2002)]. In other words, the task model is the set of tasks that users need to perform with the interactive system. In addition, the relationships between tasks are described with the execution order or dependencies between peer tasks. The tasks may contain attributes about their importance, their duration of execution, and their frequency of use.

For purposes here, the following definition is applied:

> A *task* is a goal, along with the ordered set of subtasks and actions that would satisfy it in the appropriate context (Schlungbaum, 1996).

This definition highlights the intertwining nature of tasks and goals. Actions are required to satisfy goals. Furthermore, the definition allows the decomposition of tasks into sub-tasks, with some ordering among the sub-tasks and actions. In order to support this definition, one needs to add the definitions for goal, action, and artefact:

> A *goal* is an intention to perform the task which is the state of an artefact based on (Schlungbaum, 1996);
>
> An *action* is any act which has the effect of changing or maintaining the state of an artefact based on (Schlungbaum, 1996);
>
> An *artefact* is an object which is essential for a task. Without this object, the task cannot be performed; the state of this artefact is usually changed in the course of the performance of a task. Artefacts are real things which exist in the context of task

performance. In business, artefacts are modeled as objects and represented in the business model. This implies a close relationship between the Task model and the business model.

These definitions derive the information that needs to be represented in a Task model. According to (Schlungbaum, 1996), the description of a task includes:

- A goal;
- A non-empty set of actions or other tasks which are necessary to achieve the goal;
- A plan of how to select actions or tasks;
- A model of an artifact, which is influenced by the task.

Consequently, the development of the Task model and the Domain model is interrelated. One of the goals of model-based approaches is to support user-centered interface design. Therefore, they must enable the UI designer to create the various Task models. Three other models (Dialog, Presentation, and Layout) have the Domain and Task models as inputs.

5.1.3 Dialog model

Dialog model enables one to provide dialog styles to perform tasks and to provide proven techniques for the dialog. The Dialog model defines the navigational structure of the UI. It is a more specific model and can be derived mostly from the more abstract Task, and Domain models.

A dialog model is used to describe the human-computer interaction. It specifies when the end-user can invoke commands, functions, and interaction media, when the end-user can select or specify inputs, and when the computer can query the end-user and present information (Puerta, 1997) and (Sinnig, 2004). The Dialog model describes the sequencing of input tokens, output tokens, and the way in which they are interleaved. It describes the syntactical structure of human-computer interaction. The input and output tokens are lexical elements. Therefore, and in particular, this model specifies the user commands, interaction techniques, interface responses, and command sequences permitted by the interface during user sessions. Two other models, Presentation and Layout, have the Domain, Task, and Dialog models as inputs.

5.1.4 Presentation model

The Presentation Model describes the visual appearance of the UI (Schlungbaum, 1996). This model exists at two levels of abstraction: the abstract and the concrete. In practice, they define the appearance and the form of presentation of a system within an interactive system providing solutions on how the contents or related services can be visually organized into working surfaces, the effective layout of multiple information spaces and the relationship between them. Moreover, they define the physical and logical layout suitable for specific interactive systems such as home pages, lists, and tables.

A Presentation model describes the constructs that can appear on an end-user's display, their layout characteristics, and the visual dependencies among them. The displays of most systems consist of a static part and a dynamic part. The static part includes the presentation of the standard widgets like buttons, menus, and list boxes. Typically, the static part remains fixed during the runtime of the interactive system, except for state changes like enable/disable, visible/invisible. The dynamic part displays system-dependent data, which typically change during runtime (e.g. the system generates output information, while the end-user constructs system-specific data).

The former provides an abstract view of a generic interface, which represents corresponding Task and Dialog models. Another model, Layout, has the Domain, Task, Dialog, and Presentation models as inputs.

5.1.5 Layout model

A Layout model constitutes a concrete instance for an interface. It consists of a series of UI components which defines the visual layout of a UI and the detailed dialogs for a specific platform and context of use. There may be many concrete instances of a Layout model which can be derived from Presentation and Dialog models.

The layout model makes it possible to provide conceptual models and architectures for organizing the underlying content across multiple pages, servers, databases, and computers. It is concerned with the look and feel of interactive systems and with the construction of a general drawing area (e.g. a canvas widget); and all the outputs inside a canvas must be programmed using a general-purpose programming language and a low-level graphical library.

5.2 Model Transformation

Model transformation is the process of converting one or more models – called source models – to an output model – the target model – of the same system. Transformations may combine elements of different source models in order to build a target model. Transformation rules apply to all the types of models listed above.

The following steps make up the list of transformation rules suggested in (INTERACT, 1999) and are considered as part of POMA architecture:

1. Maintain tracking structures of all class instances where needed;
2. Maintain tracking structures for association populations where needed;
3. Support state machine semantics;
4. Enforce Event ordering;
5. Preserve Action atomicity;
6. Provide a transformation for all analysis elements, including:
    - Domain, Domain Service;
    - Class, Attribute, Association, Inheritance, Associative Class, Class Service;
    - State, Event, Transition, Superstate, Substate;
    - All action-modeling elements.

The transformations between models (Si Alhir, 2003) provide a path which enables the automated implementation of a system to be derived from the various models defined for it.

5.3 Source code generation

Source code generation is not taken into account in this research project. However, the generation phase in POMA must define the source code generation rules, which will be used to generate the source code for the whole interactive system in various languages for various specific platforms.

5.4 Summary of chapter

This chapter has focused on an architectural model that combines two key approaches: model-driven and pattern-oriented. The research project has proposed five categories of models (Domain model, Task model, Dialog model, Presentation model and Layout model) to address some of the challenging problems such as: (a) decoupling the various aspects of interactive systems such as business logic, user interface, navigation, and information architecture; (b) isolating platform-specific problems from the concerns common to all interactive systems.

In chapter 6, an exploratory case study is presented to illustrate and clarify the core ideas of POMA architecture and of its practical relevance.

CHAPTER 6

CASE STUDY

6.1 Overview

This section presents a case study that describes the design of a functional user interface simplified prototype of an 'Environmental Management Interactive System' (IFEN), illustrating and clarifying the core ideas of the POMA approach and its practical relevance.

This environmental management interactive system permits requirements analysis of the environment, its evolution and its economic and social dimensions, and proposes indicators of performance. The main objectives of environmental management are the treatment and distribution of water, improving air quality, monitoring noise, the treatment of waste, the health of fauna and flora, land use, preserving coastal and marine environments, and managing natural and technological risks (IFEN).

A simplified prototype of the 'Environmental Management Interactive System' is developed here. The interactive system and corresponding models will not be tailored to different platforms. This prototype illustrates how patterns are used to establish the various models, as well as the transformation of one model into another while respecting the pattern composition rules described in section 4.1.2, the pattern mapping rules described in section 4.1.3 and the transformation rules described in section 5.2.

This case study presents a general overview of the PIM and PSM models of the 'Environmental Management Interactive System' by applying pattern composition steps and mapping rules, as well as transformation rules for the five models. The details of this illustrative case study are presented in this chapter in which the five models representing the same interactive system are illustrated on a laptop platform and on a PDA platform. The five models include the Domain model, Task model, Dialog model, Presentation model and Layout model of POMA architecture. Table 6.1 lists the patterns that will be used by the interactive system.

A prototype of a multi-platform interactive system for POMA architecture is implemented. A prototype is implemented in Java language using the Eclipse tool. There is a screenshot of the final layout of the 'Environmental Management Interactive System' illustrated in Figure 6.24. The key features of the current version of this interactive system prototype are the following:

- Support for well-arranged graphical specifications of hierarchy of POMA networks. This is achieved by the notion of a so-called tree explorer, in which the hierarchy of networks can be easily viewed and managed;
- Support for checking the correctness of network dependencies at the syntactic level. The editor contains a list of inputs and an output port for each network in the hierarchy and gives the user help to bind the right subsystem ports to the higher ports in the network hierarchy;
- Together with architectural compatibility checking, the prototype will allow one to easily define new POMA models by composing and mapping patterns which have already been defined and formalized.

Table 6.1

Pattern Summary

Pattern Name	Model Type	Problem
Login	Domain	The user's identity needs to be authenticated in order to be allowed access to protected data and/or to perform authorized operations.
Multi-Value Input Form (Seffah and Gaffar, 2006)	Domain	The user needs to enter a number of related values. These values can be of different data types, such as ‘‘date’’, ‘‘string’’, or ‘‘real’’.
Submit	Domain	The user needs to submit coordinates to the authentication process to access the system.
Feedback	Domain	The user needs help concerning the use of the Login Form.
Close	Domain	The need to close the system from the Login form.
Find (Search, Browse, Executive Summary) (Seffah and Gaffar, 2006)	Task	The need to find indicators related to the task concerned, to find environmental patterns related to the indicators, and to find a presentation tool to display the results of the indicators and the environmental patterns.
Path (Breadcrumb)	Task	The need to construct and display the path that combines the data source, task, and/or subtask.
Index Browsing	Task	The need to display all indicators listed as index browsing to navigate and select the desired ones.
Adapter	Task	The need to convert the interface of a class into another interface that clients expect; an adapter lets classes work together which could not otherwise do so because of interface incompatibility.
Builder	Task	The need to separate the construction of a complex object from its representation, so that the same construction process can create different representations.
List	Task	The need to display the information using forms.
Table	Task	The need to display the information in tables.
Map	Task	The need to display the information in geographic maps.
Graph	Task	The need to display the information in graphs.
Home Page	Task	The need to define the layout of an interactive system home page, which is important because the home page is the interactive system interface with the world and the starting point for most user visits.
Wizard (Welie, 2004) and (Sinnig, 2004)	Dialog	The user wants to achieve a single goal, but several decisions and actions need to be taken consecutively before the goal can be achieved.
Recursive Activation (Seffah and Gaffar, 2006)	Dialog	The user wants to activate and manipulate several instances of a dialog view.
Unambiguous Format (Seffah and Gaffar, 2006)	Presentation	The user needs to enter data, but may be unfamiliar with the structure of the information and/or its syntax.
Form (Seffah and Gaffar, 2006)	Presentation	The user must provide structural textual information to the system. The data to be provided are logically related.
House (Seffah and Gaffar, 2006)	Layout	Usually, the system consists of several pages/windows. The user should have the impression that it all “hangs together” and looks like one entity.

Figure 6.1 shows the graphical representation of the pattern which is used to exemplify the pattern in this case study.

Figure 6.1 Graphical representation of the pattern.

6.2 Defining the Domain Model

Acting in the horizontal line of the POMA architecture (Figure 3.2), this model is composed of two types of sub-models, [POMA.PIM]-independent Domain sub-model and [POMA.PSM]-specific Domain sub-model.

The [POMA.PIM]-independent Domain sub-model (Figure 6.2) is obtained by composing patterns and applying the composition rules.

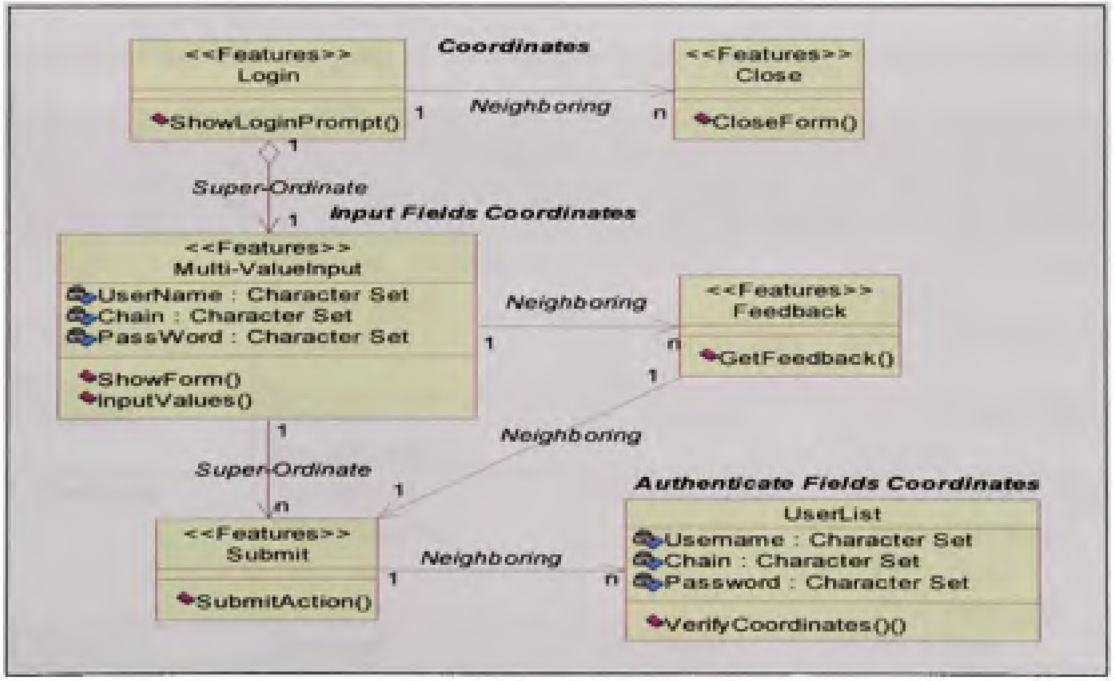

Coordinates
<<Features>>
Login
ShowLoginPrompt()
1
Neighboring
n
<<Features>>
Close
CloseForm()
1
Super-Ordinate
1
Input Fields Coordinates
<<Features>>
Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
Neighboring
1
n
<<Features>>
Feedback
GetFeedback()
1
Neighboring
1
Super-Ordinate
1
n
Authenticate Fields Coordinates
<<Features>>
Submit
SubmitAction()
Neighboring
1
n
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()()

Figure 6.2 UML class diagram of the PIM Domain model.

The following example shows the composition of a "**Close**" pattern in XML language:

```
    /* XML
    <!xml version="1.0" >
    <d-class name="Close"
        ....
    Compose-to="xml.Jbutton">
        ....
</d-class>
</xml>
```

The [POMA.PSM]-specific Domain sub-model (Figure 6.3 and Figure 6.4) is obtained by mapping composed patterns and applying the mapping rules (Table 6.2). This latter model would be used to generate the interactive system's source code by taking into account the generation code rules for a Microsoft platform.

Table 6.2 shows the mapping rules for the Domain model patterns for a laptop and PDA platforms.

Table 6.2

Example of pattern mapping of the Domain model for laptop and PDA platforms

Patterns of Microsoft Platform	Type of Mapping	Replacement patterns for Laptop platform	Replacement patterns for PDA platform
P1. Login	Identical, Scalable, or Fundamental	P1. Login	P1.s Login (small interface)
P2. Multi-value Input	Identical	P2. Multi-value Input	P2. Multi-value Input
P3. Submit	Scalable or fundamental	P3. Submit	P3.s Submit (Smaller button)
P4. Feedback	Scalable or Fundamental	P4. Feedback	P4. Feedback (Less items per page) P4.1. Next P4.2. Previous
P5. Close	Identical	P5. Close	P5. Close

Therefore, the mapped domain model is obtained. An example of the mapping of a **"Close"** pattern in Java language follows:

```
          /* Java
          <d-class name="Close"
               ....
          Maps-to="javax.swing.Jbutton">
               ....
       </d-class>
```

After the mapping, the PSM Domain model is obtained for a laptop platform – Figure 6.3 and for a PDA platform – Figure 6.4.

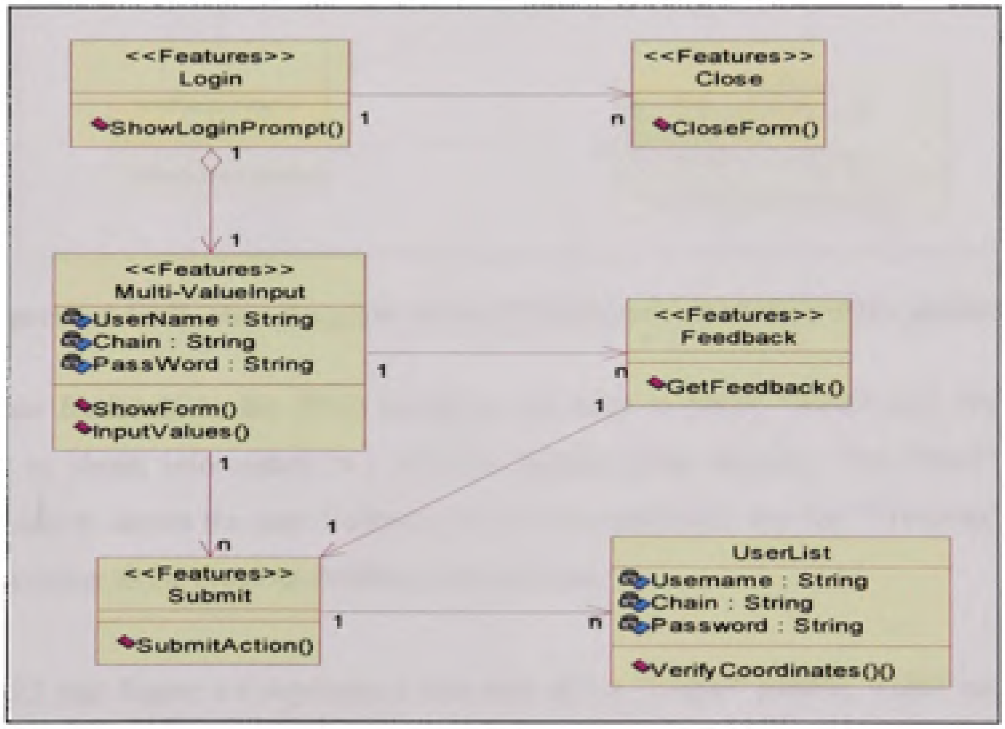

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : String
Chain : String
PassWord : String
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
<<Features>>
Submit
SubmitAction()
UserList
Username : String
Chain : String
Password : String
VerifyCoordinates()()
1
n
1
1
1
n
1
1
n
1
1
n

Figure 6.3 UML class diagram of the PSM Domain model for a laptop platform.

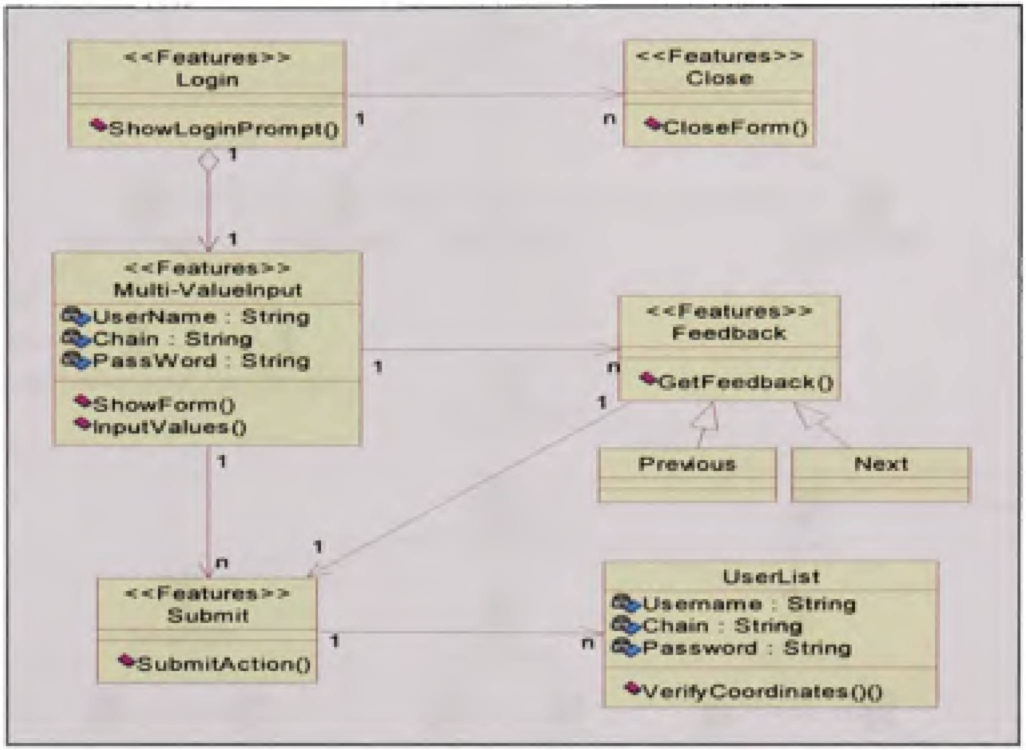

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : String
Chain : String
PassWord : String
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
Previous
Next
<<Features>>
Submit
SubmitAction()
UserList
Username : String
Chain : String
Password : String
VerifyCoordinates()()
1
n
1
1
1
n
1
1
n
1
1
n

Figure 6.4 UML class diagram of the PSM Domain model for PDA platform.

To obtain feedback on the PDA platform, we need to insert "**Next**" and "**Previous**" patterns to obtain information in a number smaller portal displays. The "**Next**" pattern enables one to access the next feedback information available, and the "**Previous**" pattern allows a return to the previous feedback information.

Figure 6.5 and Figure 6.6 represent a structure of the "**Login**" pattern, which enables the user to identify himself or herself in order to access secure or protected data and/or to perform authorized operations.

Figure 6.7 and Figure 6.8 represent an implementation of the "**Login**" pattern.

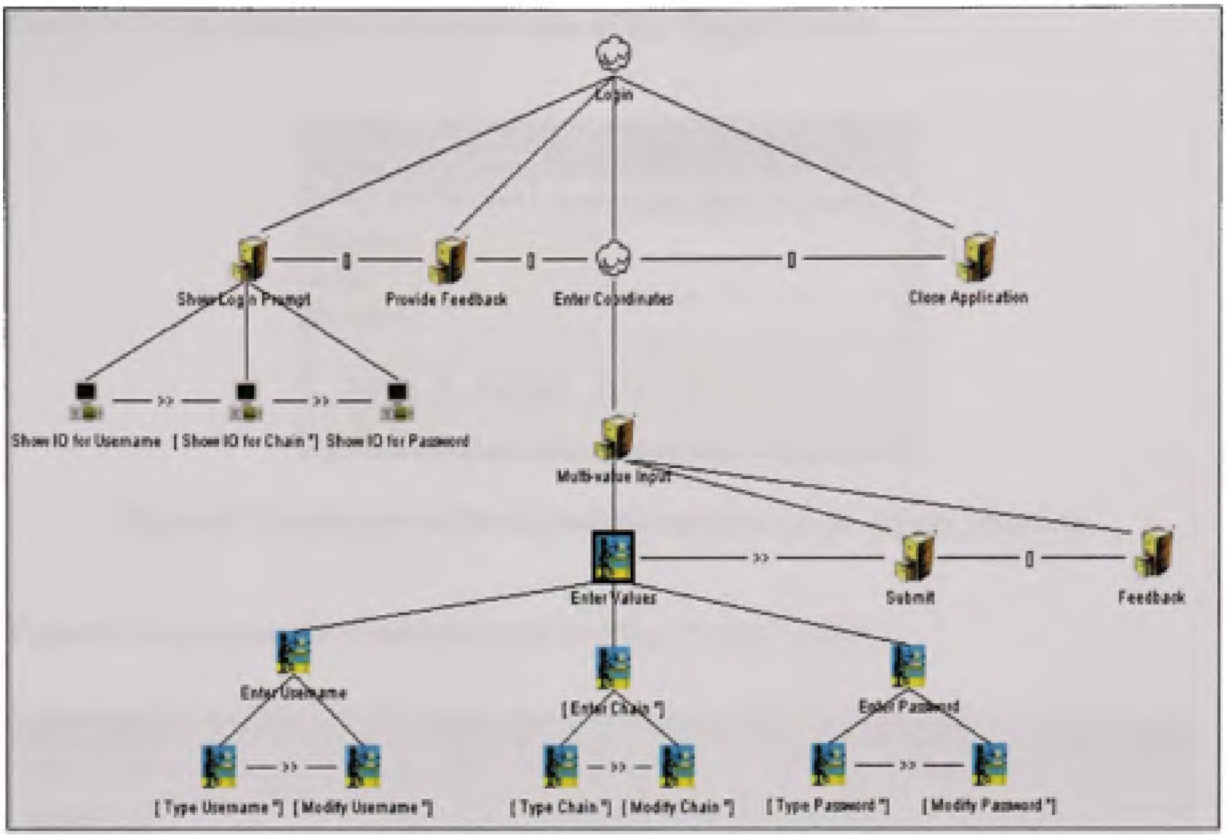

Login
Show Login Prompt
[]
Provide Feedback
[]
Enter Coordinates
[]
Close Application
Show IO for Username
>>
[Show IO for Chain *]
>>
Show IO for Password
Multi-value Input
Enter Values
>>
Submit
[]
Feedback
Enter Username
[Enter Chain *]
Enter Password
[Type Username *]
>>
[Modify Username *]
[Type Chain *]
>>
[Modify Chain *]
[Type Password *]
>>
[Modify Password *]

Figure 6.5 The *Login* pattern on the laptop platform.

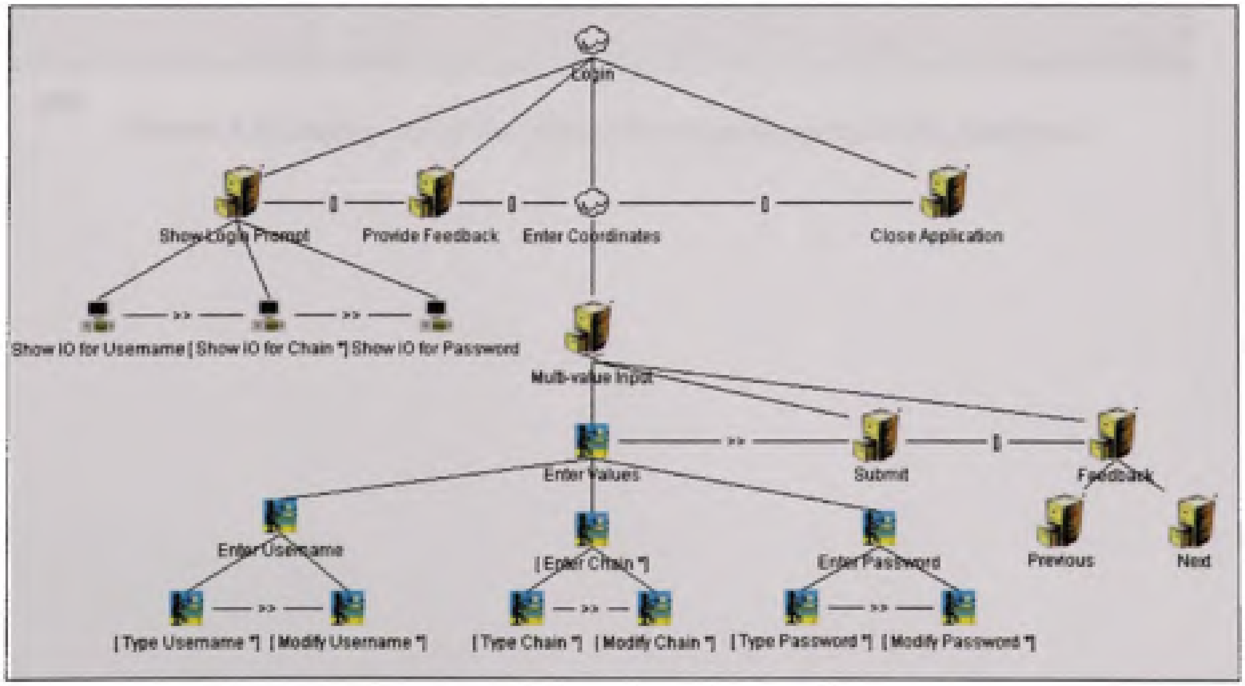

Login
Show Login Prompt
[]
Provide Feedback
[]
Enter Coordinates
[]
Close Application
Show IO for Username
>>
[Show IO for Chain *]
>>
Show IO for Password
Multi-value Input
Enter Values
>>
Submit
[]
Feedback
Previous
Next
Enter Username
[Enter Chain *]
Enter Password
[Type Username *]
>>
[Modify Username *]
[Type Chain *]
>>
[Modify Chain *]
[Type Password *]
>>
[Modify Password *]

Figure 6.6 The *Login* pattern on the PDA platform.

Figure 6.7 is an example of implementation of the **"Login"** Pattern.

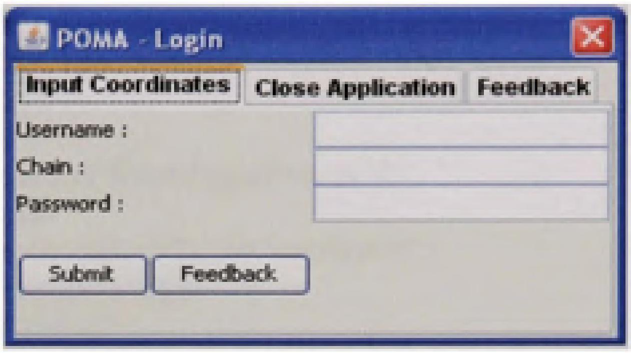
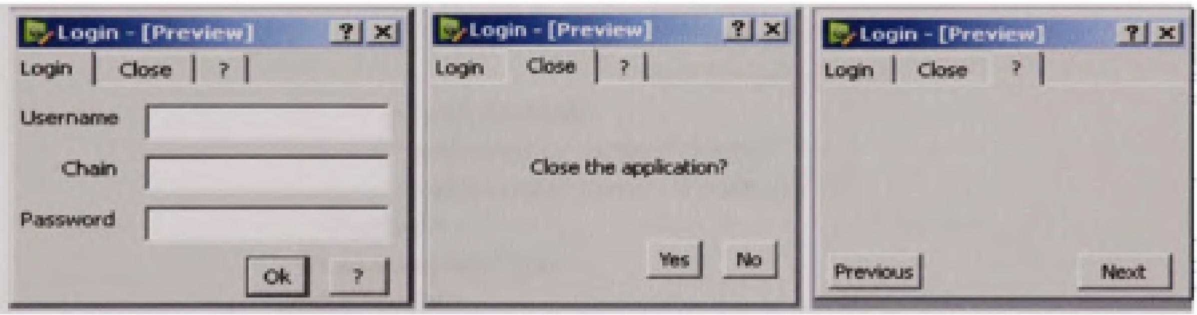

POMA - Login
Input Coordinates
Close Application
Feedback
Username :
Chain :
Password :
Submit
Feedback

Figure 6.7 Login view of the interactive system on the laptop platform.

Figure 6.8 is an example of implementation of the **"Login"** pattern.


Login - [Preview]
Login
Close
?
Username
Chain
Password
Ok
?
Login - [Preview]
Login
Close
?
Yes
No
Login - [Preview]
Login
Close
?
Previous
Next

Close the application?

Figure 6.8 Login view of the interactive system on the PDA platform.

The following is an example of the XML source code of the Domain Model for the Laptop platform of an “Environmental Management Interactive System”:

```
<?xml version=“1.0“?>
<xsd:schema xmlns:xsd=“http://www.w3.org/2007/XMLSchema“>
<xsd:group name=”Login”>
       <xsd:sequence>
            <xsd:element name=”ShowLoginPrompt”/>
               <xsd:sequence>
                   <xsd:element name=”EnterCoordinates”>
                         <xsd:complexType>
                                <xsd:all>
                                          <xsd:element name=”Multi-ValueInput”>
                                                     <xsd:complexType>
                                                               <xsd:all>
                                                                          <xsd:element name=”ShowForm”/>
                                                                          <xsd:element name=”EnterValues”>
                                                                    xsd:complexType>
                                                                           <xsd:attribute name=”Username”/>
                                                                           <xsd:attribute name=”Password”/>
                                                                   </xsd:complexType>
                                          </xsd:element>
                                          <xsd:element name=”Submit”/>
                                          <xsd:element name=”Feedback”/>
                                  </xsd:all>
                               </xsd:complexType>
                            </xsd:element>
                         <xsd:element name=”CloseApplication”/>
                     <xsd:element name=”FeedbackForLoginForm”/>
                  </xsd:all>
              </xsd:complexType>
           </xsd:element>
       </xsd:sequence>
    </xsd:sequence>
</xsd:group>
```

6.3 Defining the Task Model

After establishing the Domain model for the system in this case study, the Task model can be interactively defined. Figure 6.9 depicts the task model structure of the "Environmental Management Interactive System". Only high-level tasks and their relationships are portrayed. The overall structure and behavior of the interactive system is given. The structure provided is relatively unique for an environmental management interactive system; the concrete ''realization'' of high-level tasks has been omitted.

A large part of many interactive systems can be developed from a fixed set of reusable components. In the case of the Task model, the more those high-level tasks are decomposed, the easier it is to use the reusable task structures that have been gained or captured from other projects or systems. In this case study, these reusable task structures are documented in the form of patterns. This approach ensures an even greater degree of reuse, since each pattern can be adapted to the current use context.

The main characteristics of the environmental management system, modeled by the task structure in Figure 6.9 can be outlined as follows:

The interactive system's main functionality is accessed by logging into the system (the login task enables the management task). The key features are ''adding a guest'', which is accomplished by entering the guest's personal information and by ''selecting an environment task or subtask'' for a specific guest. The two tasks can be performed in any order. The selection process consists of four consecutively performed tasks (related through ''Enabling with Information Exchange'' operators):

1. Selecting Data Source to use;
2. Selecting Task or Subtask;
    a. Data management,
    b. Indicator management,
    c. Presentation tool management,
    d. Environmental pattern management.

Acting in the horizontal direction of the POMA architecture (Figure 3.2), this model is composed of two types of sub-model, which are: [POMA.PIM]-independent Task sub-model, and [POMA.PSM]-specific Task sub-model.

[POMA.PIM]-independent Task sub-models (Figure 6.10) are obtained by composing patterns and applying the composition rules described in the section 4.1.2.

[POMA.PSM]-specific Task sub-models (Figure 6.11) and (Figure 6.12) are obtained by mapping composed patterns and applying the mapping rules (Table 6.3). This latter model would be used to generate the system's source code by taking into account the code generation rules.

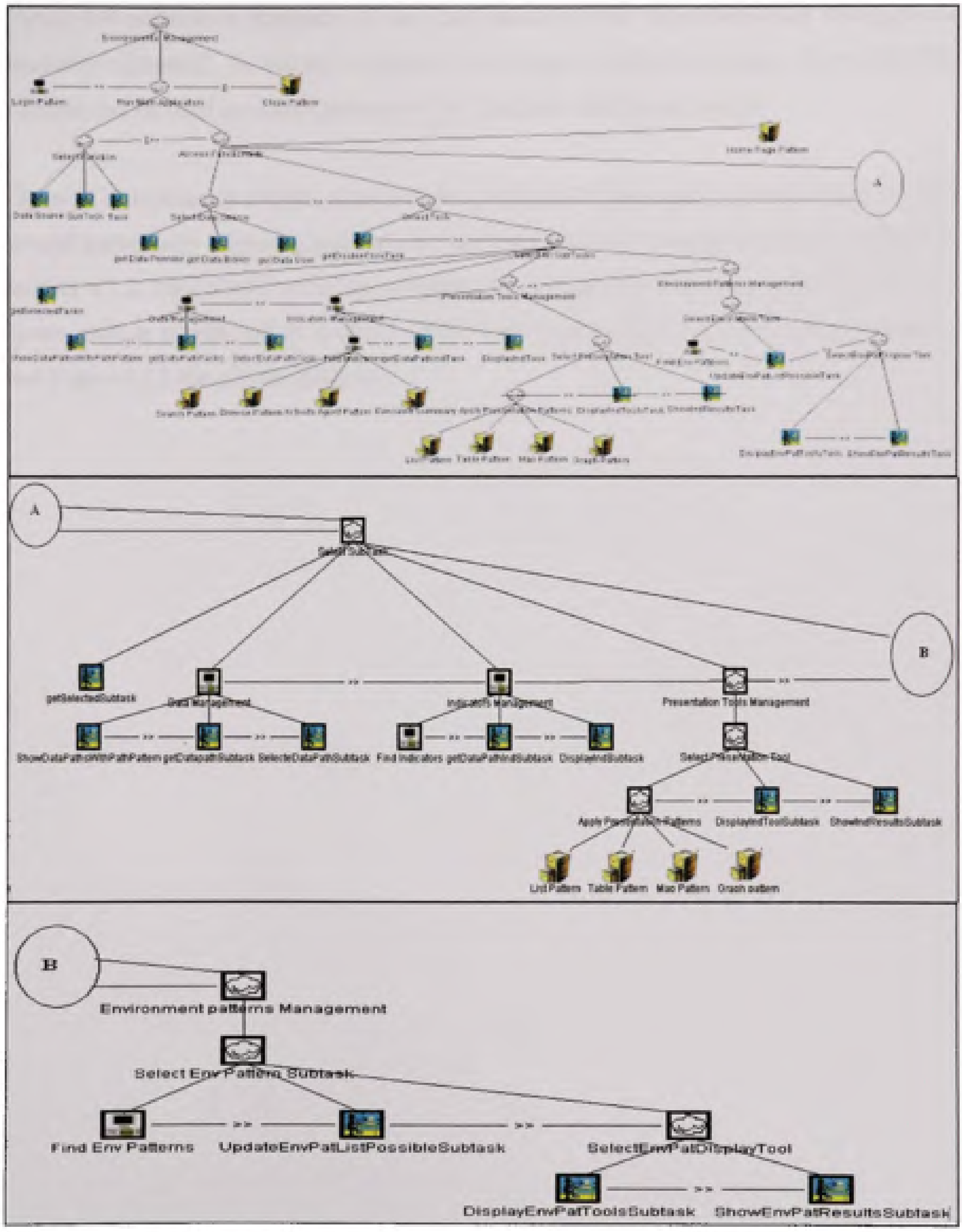

A
Home Page Pattern
A
getSelectedSubtask
Data Management
>>
Indicators Management
Presentation Tools Management
>>
B
ShowDataPathWithPathPattern
getDatapathSubtask
SelecteDataPathSubtask
Find Indicators
getDataPathIndSubtask
DisplayIndSubtask
Select Presentation Tool
Apply Presentation Patterns
DisplayIndToolSubtask
ShowIndResultsSubtask
List Pattern
Table Pattern
Map Pattern
Graph pattern
B
Environment patterns Management
Select Env Patterns Subtask
Find Env Patterns
>>
UpdateEnvPatListPossibleSubtask
>>
SelectEnvPatDisplayTool
DisplayEnvPatToolsSubtask
>>
ShowEnvPatResultsSubtask

Figure 6.9 Task model of the environmental management interactive system.

Figure 6.9 presents a structure of the Task model of the "Environmental Management Interactive System". As shown in Figure 6.9, the *Login*, *Multi-Value Input Form* and *Find* patterns can be used in order to complete the Task model at lower levels.

Figure 6.10 represents a UML class diagram of the PIM Task model, which is composed of several patterns by applying, manually by the designers, the composition rules described in section 4.1.2. This model underwent mapping by applying the mapping rules (Table 6.3) to obtain another model, which is called a PSM Task model (Figure 6.11 for a laptop platform and Figure 6.12 for a PDA platform).

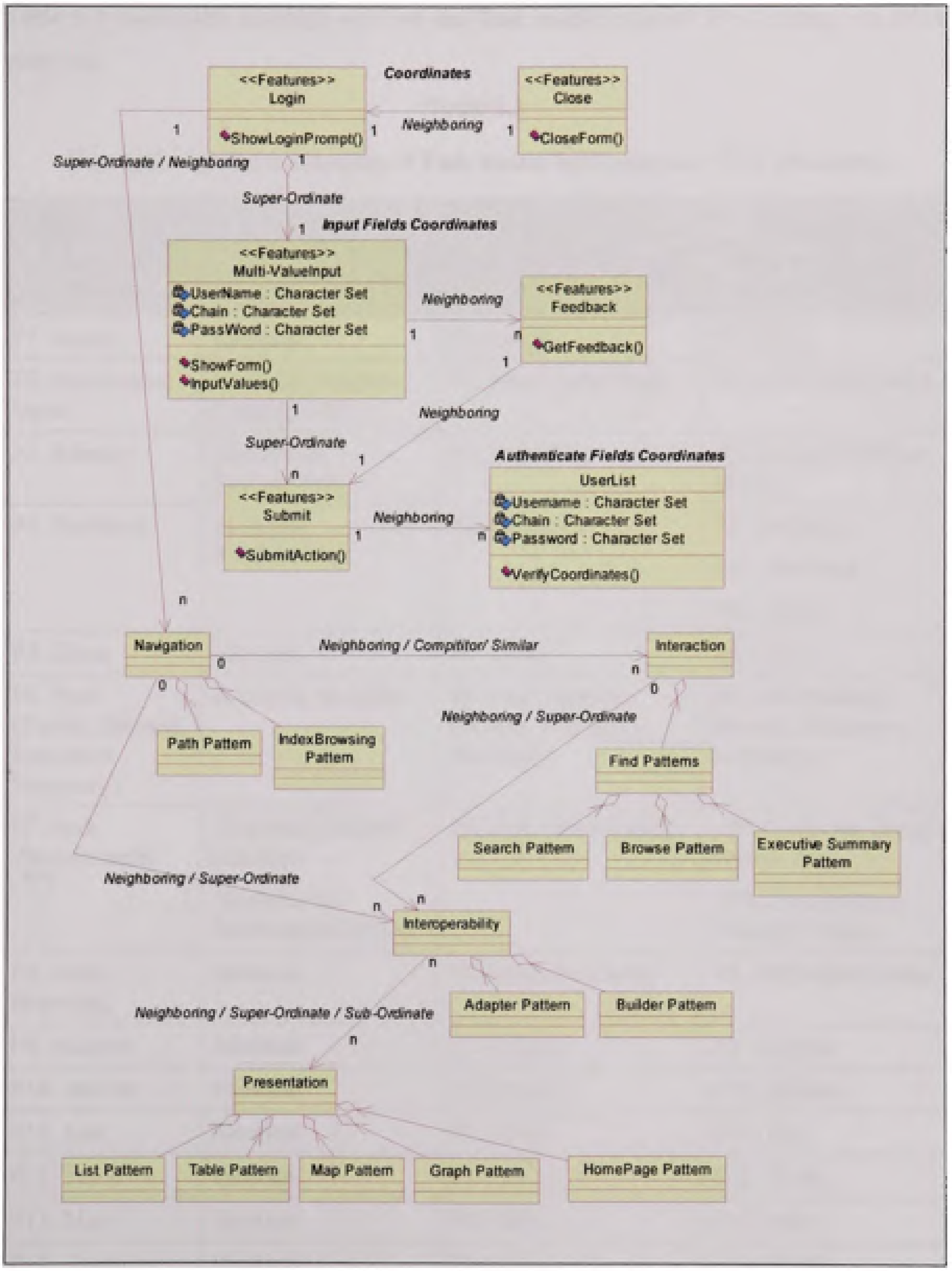

<<Features>>
Login
ShowLoginPrompt()
Coordinates
Neighboring
<<Features>>
Close
CloseForm()
1
1
1
1
Super-Ordinate / Neighboring
Super-Ordinate
1
Input Fields Coordinates
<<Features>>
Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
Neighboring
1
n
<<Features>>
Feedback
GetFeedback()
1
Neighboring
1
Super-Ordinate
1
n
Authenticate Fields Coordinates
<<Features>>
Submit
SubmitAction()
Neighboring
1
n
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
n
Navigation
Neighboring / Compititor/ Similar
0
n
Interaction
0
0
Neighboring / Super-Ordinate
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Neighboring / Super-Ordinate
n
n
Interoperability
n
Adapter Pattern
Builder Pattern
Neighboring / Super-Ordinate / Sub-Ordinate
n
Presentation
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern

Figure 6.10 UML class diagram of the PIM Task model.

Table 6.3 shows the mapping rules of the Task model patterns for a laptop and PDA platforms.

Table 6.3

Example of pattern mapping of Task model for laptop and PDA platforms

Patterns of Microsoft Platform	Type of Mapping	Replacement patterns for Laptop platform	Replacement patterns for PDA platform
P1. Login	Identical	P1. Login	P1. Login
P2. Multi-value Input	Identical, Scalable, Fundamental	P2. Multi-value Input	P2. Multi-value Input
P3. Submit	Scalable or Fundamental	P3. Submit	P3.s Submit (Smaller button)
P4. Feedback	Identical, Fundamental	P4. Feedback	P4. Feedback P4.1. Previous P4.2. Next
P5. Close	Identical	P5. Close	P5. Close
P6. Find (Search, Browse, Executive Summary)	Identical, Scalable	P6. Find (Search, Browse, Executive Summary	P6. Find (Search, Browse, Executive Summary)
P7. Path (Breadcrumb)	-Identical, Scalable (Laptop) -Scalable or fundamental (PDA)	P7. Path (Breadcrumb)	- P7.1s Shorter Bread Crumb Trial - P7.2 Drop-down "History" menu
P8. Index Browsing	Identical	P8. Index Browsing	P8. Drop-down menu
P9. Adapter	Identical	P9. Adapter	P9. Adapter
P10. Builder	Identical	P10. Builder	P10. Builder
P11. List	Identical	P11. List	P11. List
P12. Table	Identical	P12. Table	P12. Table
P13. Map	Identical	P13. Map	P13. Map
P14. Graph	Identical	P14. Graph	P14. Graph
P15. Home Page	Identical	P15. Home Page	P15. Home Page

After the mapping, the PSM Task model is obtained for a laptop platform – Figure 6.11.

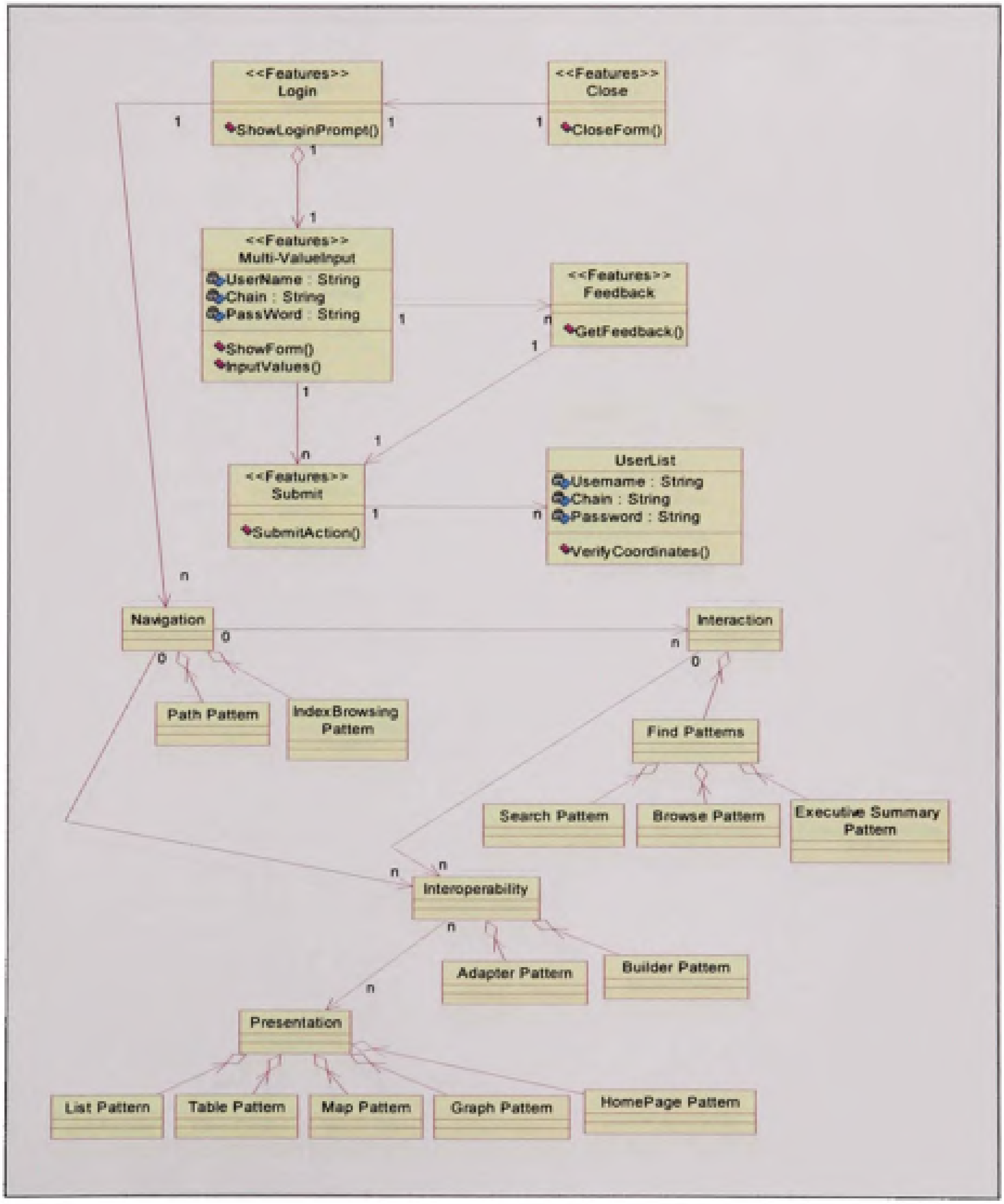

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : String
Chain : String
PassWord : String
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
<<Features>>
Submit
SubmitAction()
UserList
Username : String
Chain : String
Password : String
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Adapter Pattern
Builder Pattern
Presentation
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
1
1
1
1
1
1
1
n
1
1
1
n
1
n
n
0
n
0
0
n
n
n
n

Figure 6.11 UML class diagram of the PSM Task model mapped for a laptop platform.

After the mapping, the PSM Task model is obtained for a PDA platform – Figure 6.12.

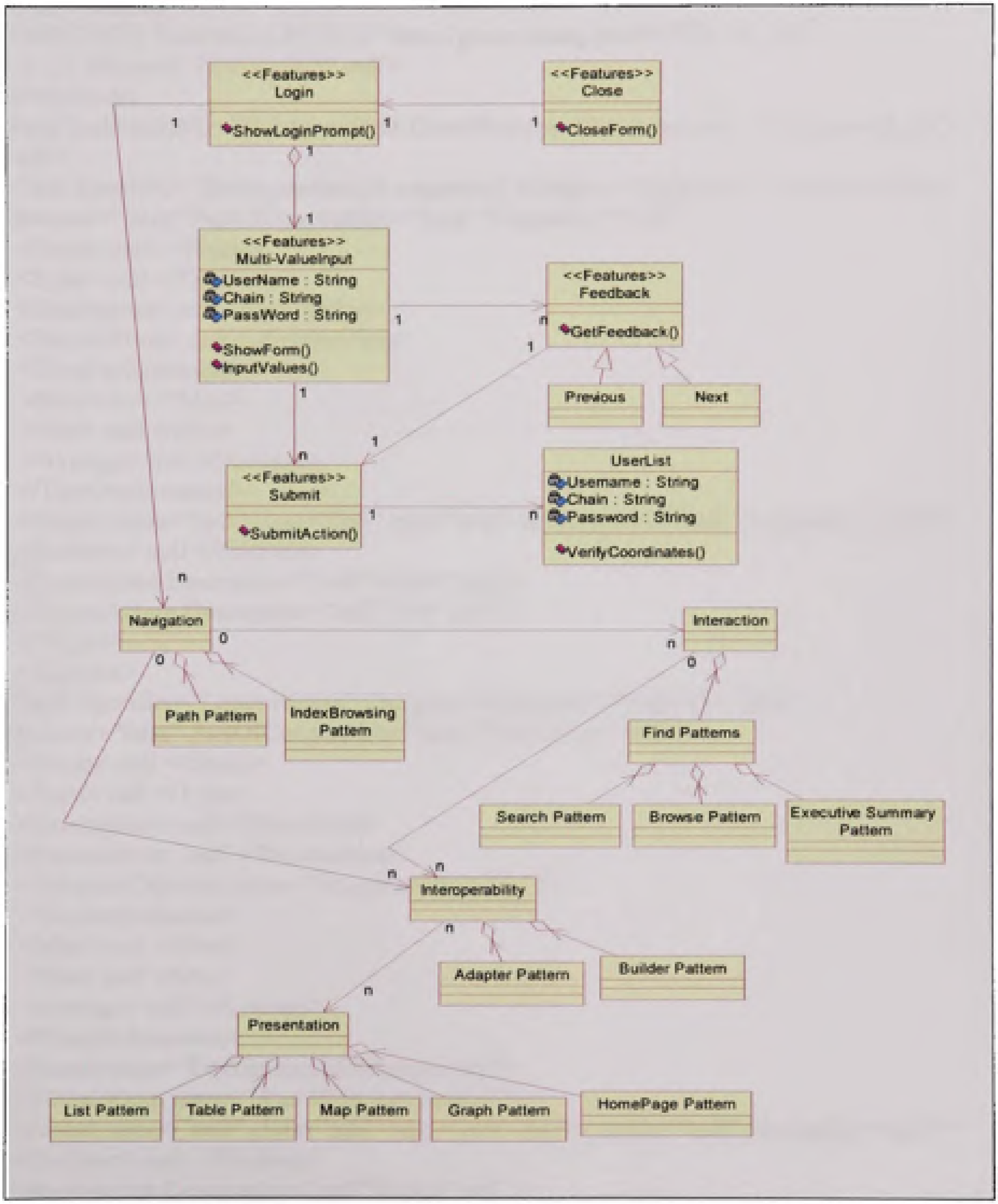

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : String
Chain : String
PassWord : String
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
Previous
Next
<<Features>>
Submit
SubmitAction()
UserList
Username : String
Chain : String
Password : String
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Adapter Pattern
Builder Pattern
Presentation
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
1
n
0

Figure 6.12 UML class diagram of the PSM Task model mapped for a PDA platform.

The following is an example of the XML source code portion of the Task Model for a Laptop platform of the “Environmental Management Interactive System”:

```
<?xml version= ‘1.0’?>
<!DOCTYPE TaskModel PUBLIC “http://giove.cnuce.cnr.it/CTTDTD.dtd”
“..\..\..\..\Teresa\CTT\CTTDTD.dtd”>
<TaskModel
NameTaskModelID=”C:\Momo\PhD\These\Prototype\Environmental_Management_CTT.
xml”>
<Task Identifier="Environmental Management” Category=”abstraction” Iterative=”false”
Optional=”false” PartOfCooperation=”false” Frequency=”null”>
  <Name> null </Name>
  <Type> null </Type>
  <Description> null </Description>
  <Precondition> null </Precondition>
  <TimePerformance>
   <Max> null </Max>
   <Min> null </Min>
   <Average> null </Average>
  </TimePerformance>
  <Object name="null" class="null" type="null" access_mode="null" cardinality="null">
  <Platform> null </Platform>
  <InputAction Description=”null” From=”null”/>
  <OutputAction Description=”null” To=”null”/>
  </Object>
  <SubTask>
<Task Identifier=”Login Pattern” Category=”application” Iterative=”false”
Optional=”false” PartOfCooperation=”false” Frequency=”null”>
  <Name> null </Name>
  <Type> null </Type>
  <Description> null </Description>
  <Precondition> null </Precondition>
  <TemporalOperator name="SequentialEnabling"/>
  <TimePerformance>
   <Max> null </Max>
   <Min> null </Min>
   <Average> null </Average>
  </TimePerformance>
  <Parent name=”Environmental Management”/>
  <SiblingRight name="Run Main Application”/>
  <Object name="null" class="null" type="null" access_mode="null" cardinality="null">
  <Platform> null </Platform>
  <InputAction Description=”null” From=”null”/>
  <OutputAction Description=”null” To=”null”/>
  </Object>
```

```
 </Task>
  …
 <SubTask>
<Task Identifier="ShowDataPathsWithPathPattern" Category="interaction"
Iterative="false" Optional="false" PartOfCooperation="false" Frequency="null">
 <Name> null </Name>
 <Type> null </Type>
 <Description> null </Description>
 <Precondition> null </Precondition>
 <TemporalOperator name="SequentialEnabling"/>
 <TimePerformance>
  <Max> null </Max>
  <Min> null </Min>
  <Average> null </Average>
 </TimePerformance>
 <Parent name="Data Management"/>
 <SiblingRight name="getDataPathTasks"/>
 <Object name="null" class="null" type="null" access_mode="null" cardinality="null">
 <Platform> null </Platform>
 <InputAction Description="null" From="null"/>
 <OutputAction Description="null" To="null"/>
 </Object>
 </Task>
  …
 </SubTask>
 </Task>
</TaskModel>
```

6.4 Defining the Dialog Model

Acting in the horizontal line of the POMA architecture (Figure 3.2), the Dialog model is composed of two types of sub-model, [POMA.PIM]-independent Dialog sub-model, and [POMA.PSM]-specific Dialog sub-model.

[POMA.PIM]-independent dialog sub-model (Figure 6.13) is obtained by composing patterns and applying, manually by the designers, the composition rules described in section 4.1.2.

The Wizard dialog pattern emerges as the best choice for implementation. It suggests a dialog structure where a set of dialog views is arranged sequentially and the ''last'' task of each dialog view initiates the transition to the subsequent dialog view. Figure 6.14 depicts the Wizard dialog pattern's suggested graph structure.

[POMA.PSM]-specific Dialog sub-model (Figure 6.15) is obtained by mapping composed patterns and applying the mapping rules (Table 6.4). This [POMA.PSM] model is used to generate the interactive system's source code by taking into account the code generation rules.

Figure 6.13 represents a UML class diagram of the PIM Dialog model, which is composed of several patterns. This model underwent mapping by applying the mapping rules (Table 6.4) to obtain another model, which is called PSM Dialog model (Figure 6.15 for a laptop platform and Figure 6.16 for a PDA platform).

However, the sequential structure of the subtask process must be slightly modified in order to enable the user to view the details of multiple subtasks at the same time. Specifically, this behavior should be modeled using the Recursive Activation dialog pattern. This pattern is used when the user wishes to activate and manipulate several instances of a dialog view.

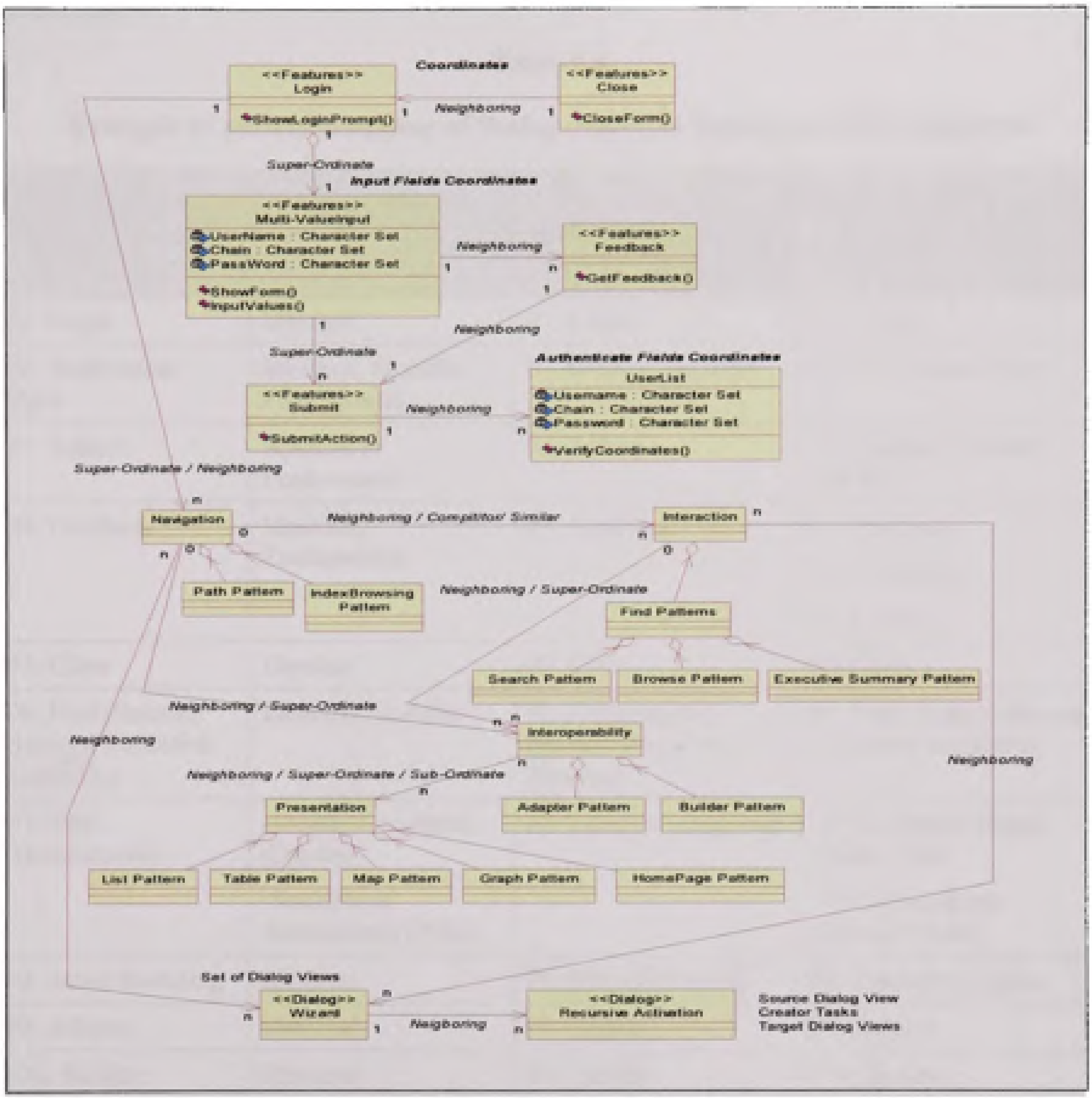

Coordinates
<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
Neighboring
1
1
1
1
Super-Ordinate
1
Input Fields Coordinates
<<Features>>
Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
Neighboring
1
n
<<Features>>
Feedback
GetFeedback()
1
Neighboring
1
Super-Ordinate
n
1
Authenticate Fields Coordinates
<<Features>>
Submit
SubmitAction()
Neighboring
1
n
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
Super-Ordinate / Neighboring
n
Navigation
Neighboring / Compititor/ Similar
Interaction
n
n
0
0
n
0
Path Pattern
IndexBrowsing Pattern
Neighboring / Super-Ordinate
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Neighboring / Super-Ordinate
n
n
Interoperability
Neighboring
n
Neighboring / Super-Ordinate / Sub-Ordinate
n
Presentation
Adapter Pattern
Builder Pattern
Neighboring
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
Set of Dialog Views
n
<<Dialog>>
Wizard
n
1
Neigboring
n
<<Dialog>>
Recursive Activation
Source Dialog View
Creator Tasks
Target Dialog Views

Figure 6.13 UML class diagram of a PIM Dialog Model.

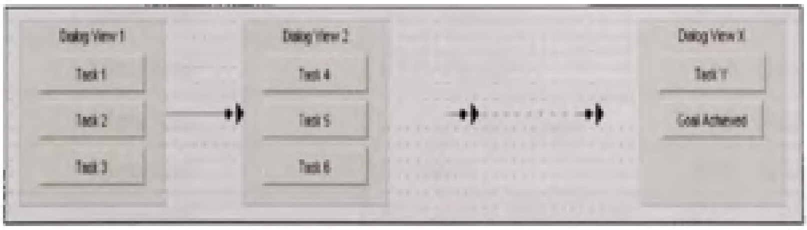

Dialog View 1
Task 1
Task 2
Task 3
Dialog View 2
Task 4
Task 5
Task 6
Dialog View X
Task Y
Goal Achieved

Figure 6.14 Graph structure suggested by the Wizard pattern.

Table 6.4 shows the mapping rules of the Dialog model patterns for laptop and PDA platforms.

Table 6.4

Example of pattern mapping of Dialog model for laptop and PDA platforms

Patterns of Microsoft Platform	Type of Mapping	Replacement patterns for Laptop platform	Replacement patterns for PDA platform
P1. Login	Identical	P1. Login	P1. Login
P2. Multi-value Input	Identical, Scalable, Fundamental	P2. Multi-value Input	P2. Multi-value Input
P3. Submit	Scalable or fundamental	P3. Submit	P3.s Submit (Smaller button)
P4. Feedback	Identical, Fundamental	P4. Feedback	P4. Feedback P4.1. Previous P4.2. Next
P5. Close	Identical	P5. Close	P5. Close
P6. Find (Search, Browse, Executive Summary)	Identical, Scalable	P6. Find (Search, Browse, Executive Summary	P6. Find (Search, Browse, Executive Summary)
P7. Path (Breadcrumb)	-Identical, Scalable (Laptop) -Scalable or fundamental (PDA)	P7. Path (Breadcrumb)	- P7.1s Shorter Bread Crumb Trial - P7.2 Drop-down "History" menu
P8. Index Browsing	Identical	P8. Index Browsing	P8. Drop-down menu
P9. Adapter	Identical	P9. Adapter	P9. Adapter
P10. Builder	Identical	P10. Builder	P10. Builder
P11. List	Identical	P11. List	P11. List
P12. Table	Identical	P12. Table	P12. Table
P13. Map	Identical	P13. Map	P13. Map
P14. Graph	Identical	P14. Graph	P14. Graph
P15. Home Page	Identical	P15. Home Page	P15. Home Page
P16. Wizard	Identical	Wizard	P16. Wizard
P17. Recursive Activation	Identical	Recursive Activation	P17. Recursive Activation

After the mapping, the PSM Dialog model is obtained for a laptop platform – Figure 6.15.

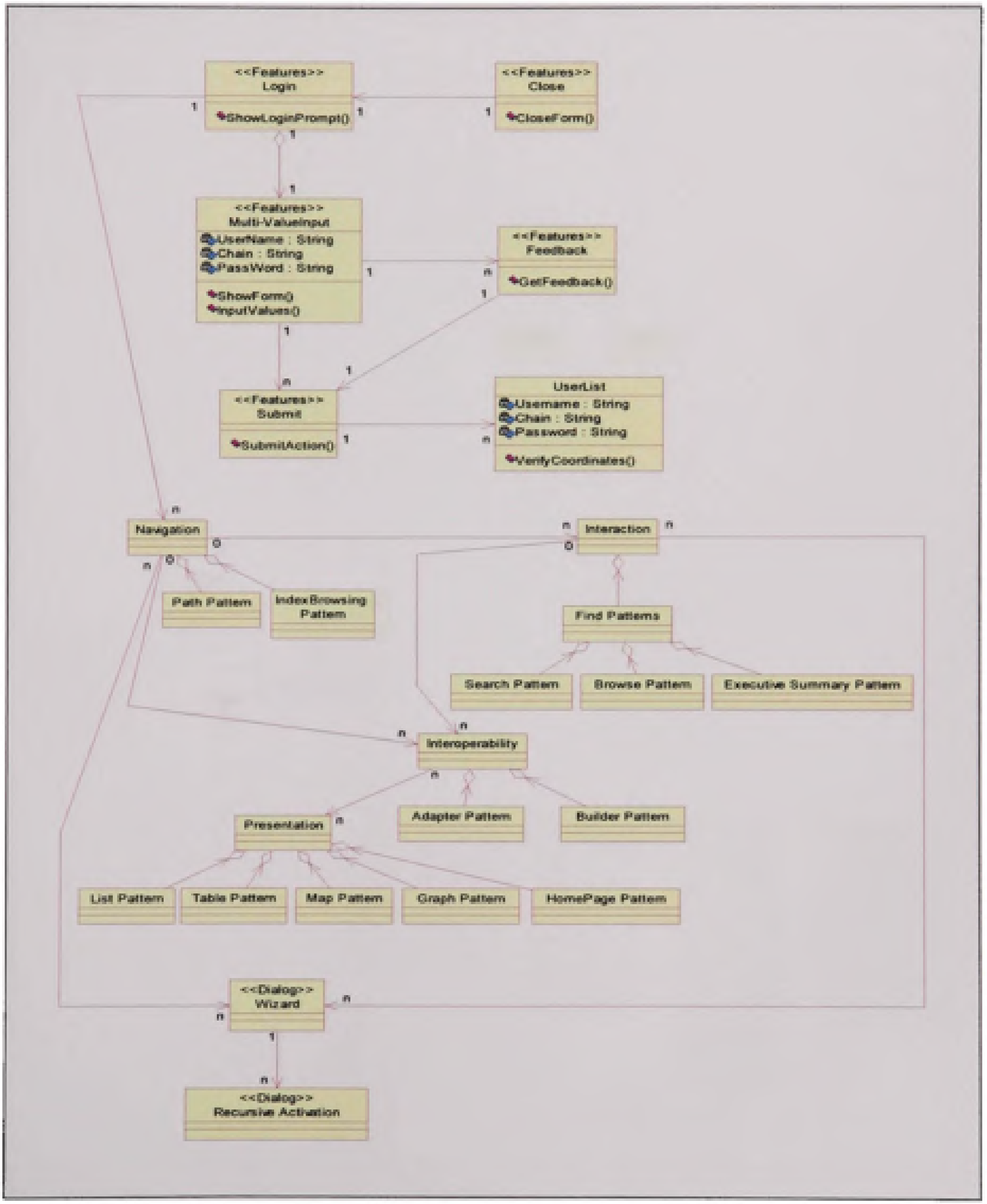

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : String
Chain : String
PassWord : String
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
<<Features>>
Submit
SubmitAction()
UserList
Username : String
Chain : String
Password : String
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Adapter Pattern
Builder Pattern
Presentation
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
<<Dialog>>
Wizard
<<Dialog>>
Recursive Activation

Figure 6.15 UML class diagram of the PSM Dialog model for a laptop platform.

After the mapping, the PSM Dialog model is obtained for a PDA platform – Figure 6.16.

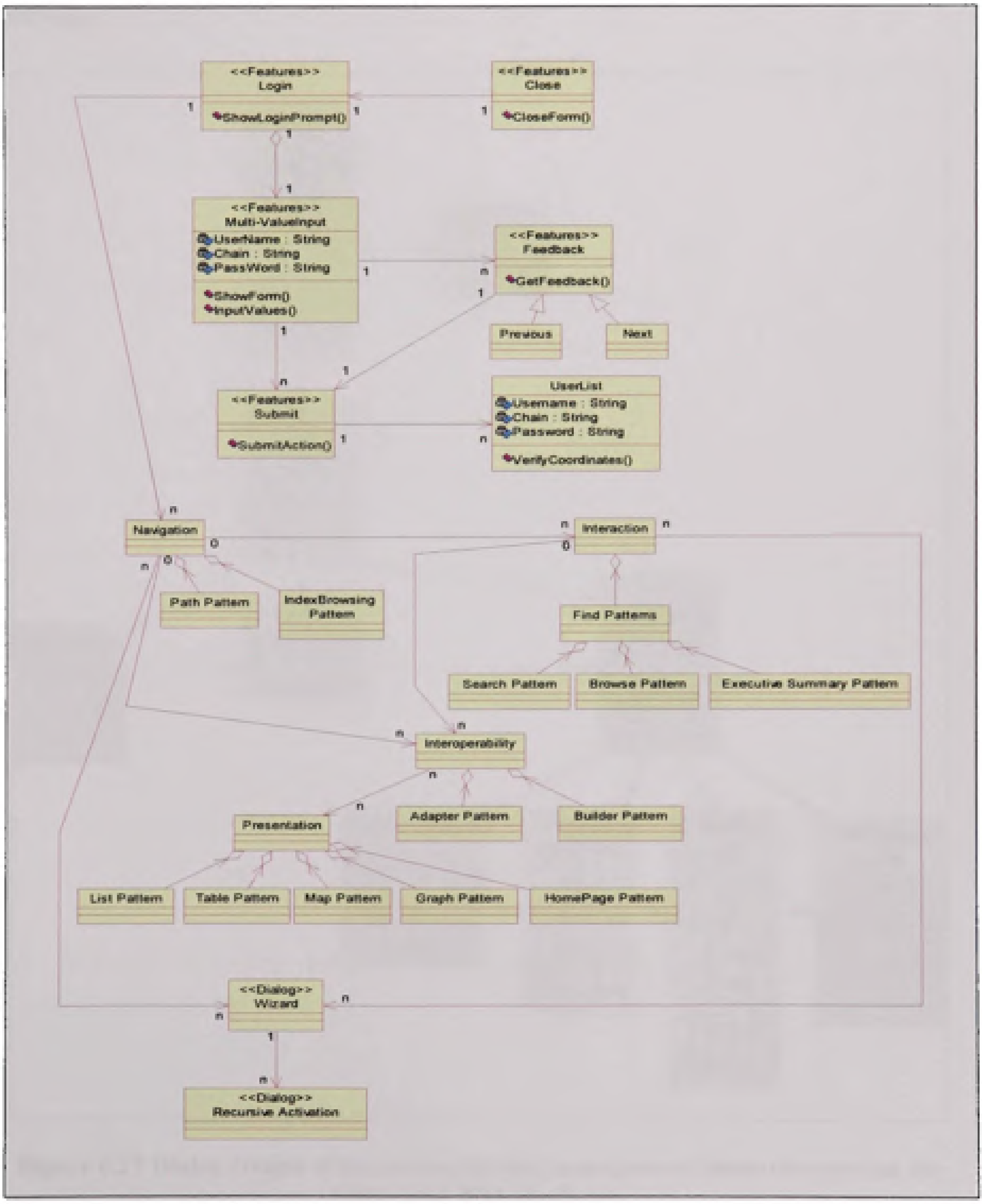

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : String
Chain : String
PassWord : String
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
Previous
Next
<<Features>>
Submit
SubmitAction()
UserList
Username : String
Chain : String
Password : String
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Presentation
Adapter Pattern
Builder Pattern
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
<<Dialog>>
Wizard
<<Dialog>>
Recursive Activation
1
n
0

Figure 6.16 UML class diagram of the PSM Dialog model for a PDA platform.

Figure 6.17 depicts the various dialog view interactions of the "Environmental Management Interactive System's" suggested dialog graph structure for laptop and PDA platforms.

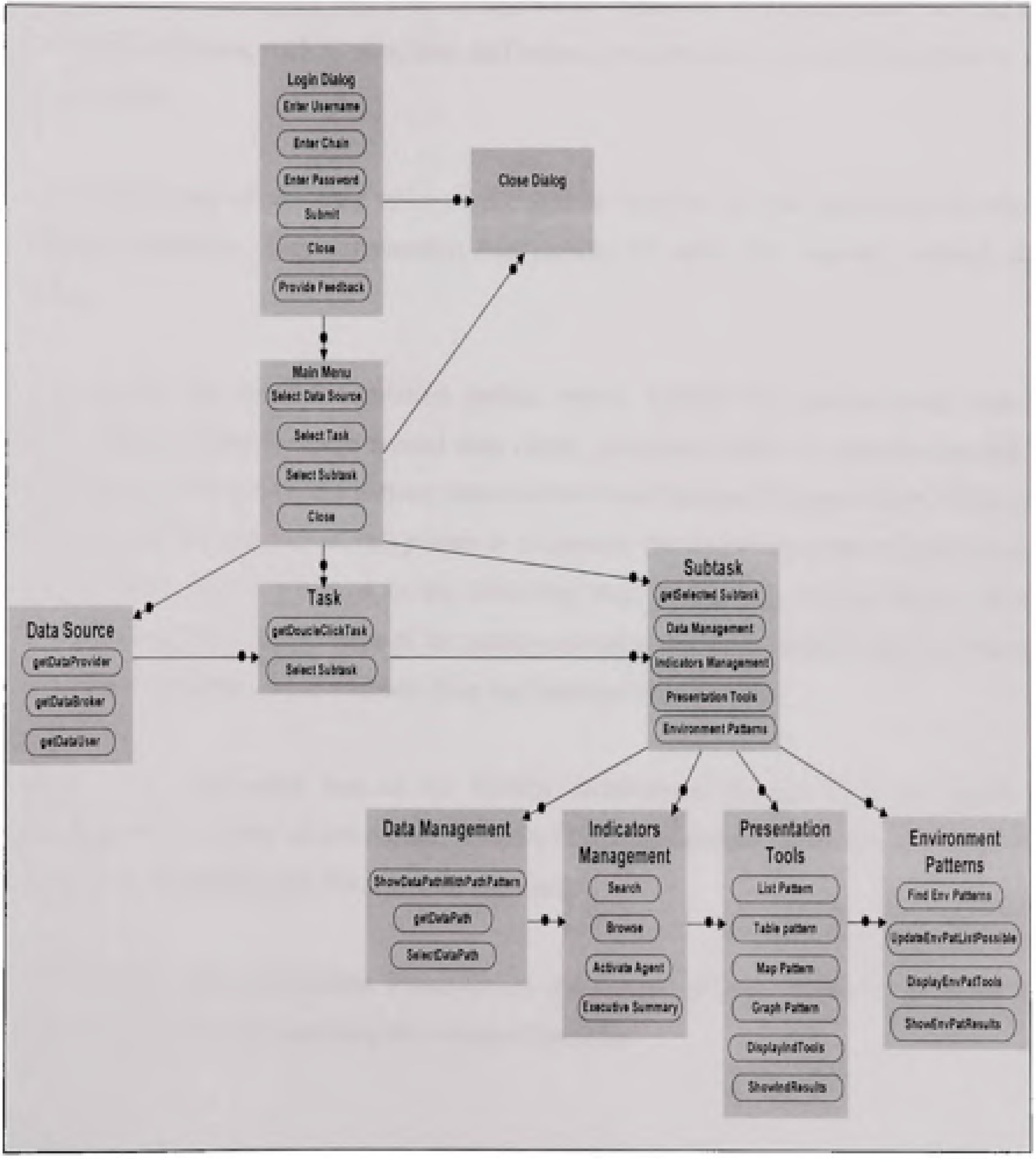

Login Dialog
Enter Username
Enter Chain
Enter Password
Submit
Close
Provide Feedback
Close Dialog
Main Menu
Select Data Source
Select Task
Select Subtask
Close
Data Source
getDataProvider
getDataBroker
getDataUser
Task
getDoubleClickTask
Select Subtask
Subtask
getSelected Subtask
Data Management
Indicators Management
Presentation Tools
Environment Patterns
Data Management
ShowDataPathWithPathPattern
getDataPath
SelectDataPath
Indicators Management
Search
Browse
Activate Agent
Executive Summary
Presentation Tools
List Pattern
Table pattern
Map Pattern
Graph Pattern
DisplayIndTools
ShowIndResults
Environment Patterns
Find Env Patterns
UpdateEnvPatListPossible
DisplayEnvPatTools
ShowEnvPatResults

Figure 6.17 Dialog Graph of the environmental management interactive system for laptop and PDA platforms.

6.5 Defining the Presentation and Layout Models

In order to define the Presentation model for this case study, the grouped tasks of each dialog view are associated with a set of interaction elements, including forms, buttons and lists. Style attributes, such as size, font, and colour, remain unset and will be defined by the layout model.

A significant part of the user tasks of the system revolves around providing structured textual information. This information can usually be split into logically related data chunks.

At this point, the Form Presentation pattern, which handles this precise issue, can be applied using a form for each related data chunk, populated with the elements needed to enter the data. Moreover, the pattern refers to the Unambiguous Format pattern which can be employed. The purpose of this pattern is to prevent the user from entering syntactically incorrect data, and is achieved in the following way: Depending on the domain of the object to be entered, the instance of the pattern provides the most suitable input interaction elements by drawing on information from the business object model.

Acting in the horizontal line of the POMA architecture (Figure 3.2), the model is composed of two types of sub-model, [POMA.PIM]-independent Presentation sub-model, and [POMA.PSM]-specific Presentation sub-model.

The [POMA.PIM]-independent Presentation sub-model (Figure 6.18) is obtained by composing patterns and applying the composition rules.

The [POMA.PSM]-specific Presentation sub-model (Figure 6.19 and Figure 6.20) is obtained by mapping composed patterns and applying the mapping rules (Table 6.5). This model is used to generate the system's source code by taking into account the code generation rules.

Figure 6.18 represents a UML diagram of the PIM Presentation model, which is composed of several patterns. This model underwent mapping by applying the mapping rules (Table 6.5) to obtain another model, which is called the PSM Presentation model (Figure 6.19 for a laptop platform and Figure 6.20 for a PDA platform).

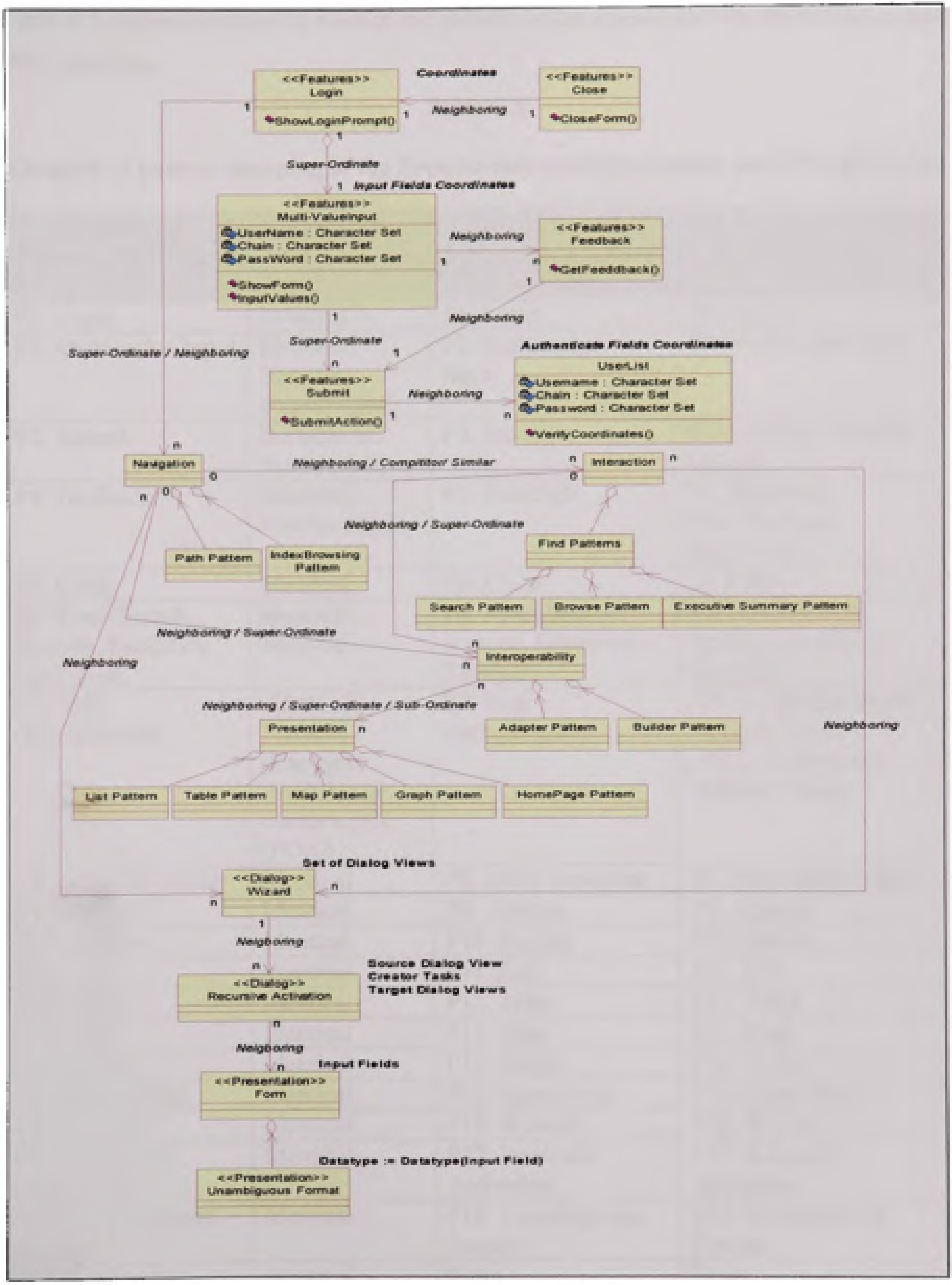

<<Features>> Login
ShowLoginPrompt()
Coordinates
<<Features>> Close
CloseForm()
Neighboring
1
1
1
1
Super-Ordinate
1
Input Fields Coordinates
<<Features>> Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
Neighboring
<<Features>> Feedback
GetFeeddback()
1
n
1
Neighboring
1
Super-Ordinate
n
1
Super-Ordinate / Neighboring
Authenticate Fields Coordinates
<<Features>> Submit
SubmitAction()
Neighboring
1
n
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
n
Navigation
0
n
0
Neighboring / Compititor/ Similar
n
0
Interaction
n
Neighboring / Super-Ordinate
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Neighboring / Super-Ordinate
Neighboring
n
n
Interoperability
n
Neighboring / Super-Ordinate / Sub-Ordinate
Presentation
n
Adapter Pattern
Builder Pattern
Neighboring
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
Set of Dialog Views
<<Dialog>> Wizard
n
n
1
Neigboring
n
<<Dialog>> Recursive Activation
Source Dialog View
Creator Tasks
Target Dialog Views
n
Neigboring
n
Input Fields
<<Presentation>> Form
Datatype := Datatype(Input Field)
<<Presentation>> Unambiguous Format

Figure 6.18 UML class diagram of a PIM Presentation model.

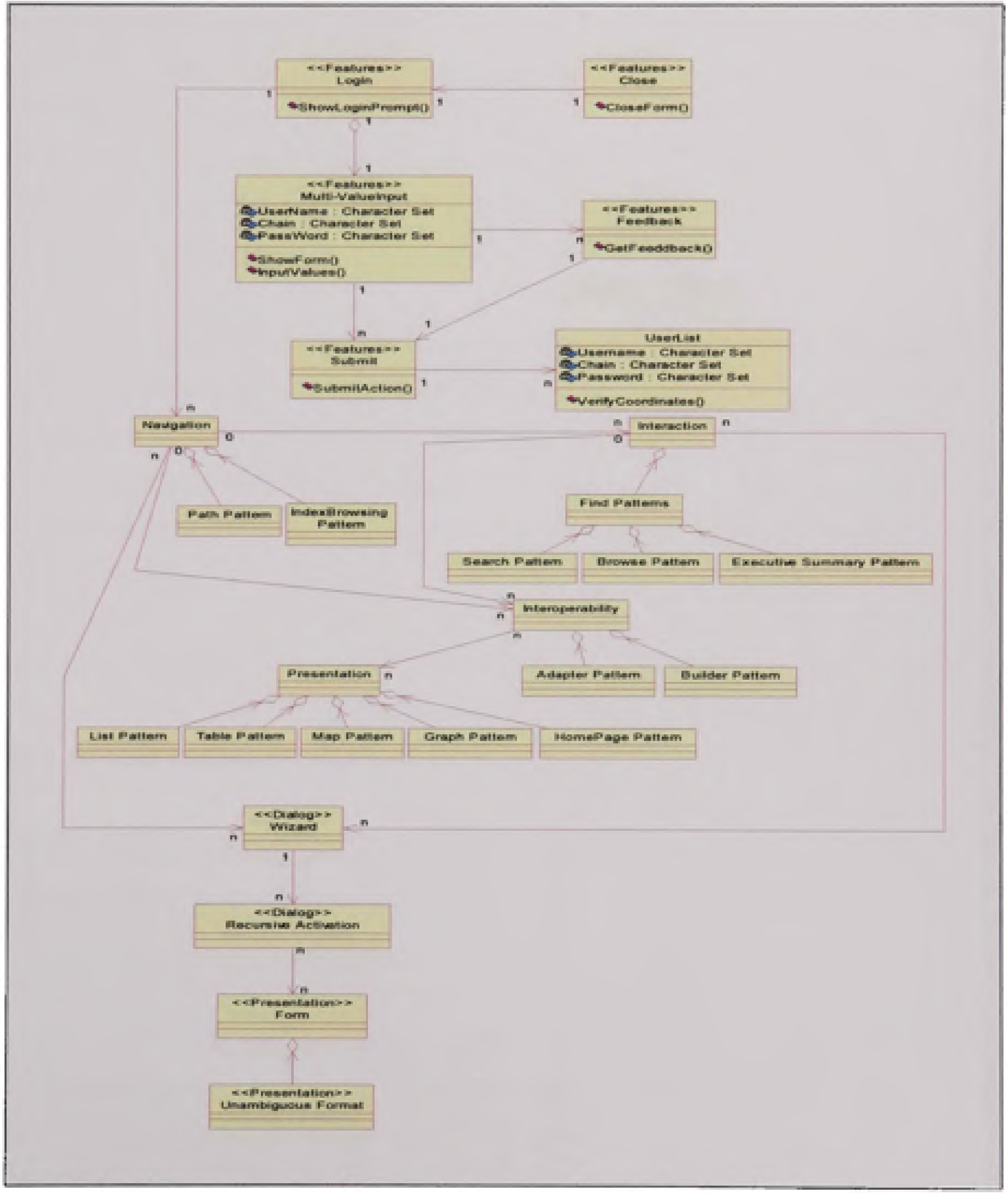
Table 6.5 shows the mapping rules of the patterns of the Presentation model for laptop and PDA platforms.

Table 6.5

Example of pattern mapping of the Presentation model for laptop and PDA platforms

Patterns of Microsoft Platform	Type of Mapping	Replacement patterns for Laptop platform	Replacement patterns for PDA platform
P1. Login	Identical	P1. Login	P1. Login
P2. Multi-value Input	Identical, Scalable, Fundamental	P2. Multi-value Input	P2. Multi-value Input
P3. Submit	Scalable or fundamental	P3. Submit	P3.s. Submit (Smaller button)
P4. Feedback	Identical, Fundamental	P4. Feedback	P4. Feedback P4.1. Previous P4.2. Next
P5. Close	Identical	P5. Close	P5. Close
P6. Find (Search, Browse, Executive Summary)	Identical, Scalable	P6. Find (Search, Browse, Executive Summary	P6. Find (Search, Browse, Executive Summary)
P7. Path (Breadcrumb)	-Identical, Scalable (Laptop) -Scalable or fundamental (PDA)	P7. Path (Breadcrumb)	- P7.1s. Shorter Bread Crumb Trial - P7.2. Drop-down "History" menu
P8. Index Browsing	Identical	P8. Index Browsing	P8. Drop-down menu
P9. Adapter	Identical	P9. Adapter	P9. Adapter
P10. Builder	Identical	P10. Builder	P10. Builder
P11. List	Identical	P11. List	P11. List
P12. Table	Identical	P12. Table	P12. Table
P13. Map	Identical	P13. Map	P13. Map
P14. Graph	Identical	P14. Graph	P14. Graph
P15. Home Page	Identical	P15. Home Page	P15. Home Page
P16. Wizard	Identical	P16. Wizard	P16. Wizard
P17. Recursive Activation	Identical	P17. Recursive Activation	P17. Recursive Activation
P18. Unambiguous Format	Identical	P18. Unambiguous Format	P18. Unambiguous Format
Form	Identical	Form	Form

After the mapping, the PSM Presentation model is obtained for a laptop platform – Figure 6.19.



<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeeddback()
<<Features>>
Submit
SubmitAction()
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Presentation
Adapter Pattern
Builder Pattern
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
<<Dialog>>
Wizard
<<Dialog>>
Recursive Activation
<<Presentation>>
Form
<<Presentation>>
Unambiguous Format
1
n
0

Figure 6.19 UML class diagram of the PSM Presentation model for a laptop platform.

After the mapping, the PSM Presentation model is obtained for a PDA platform – Figure 6.20.

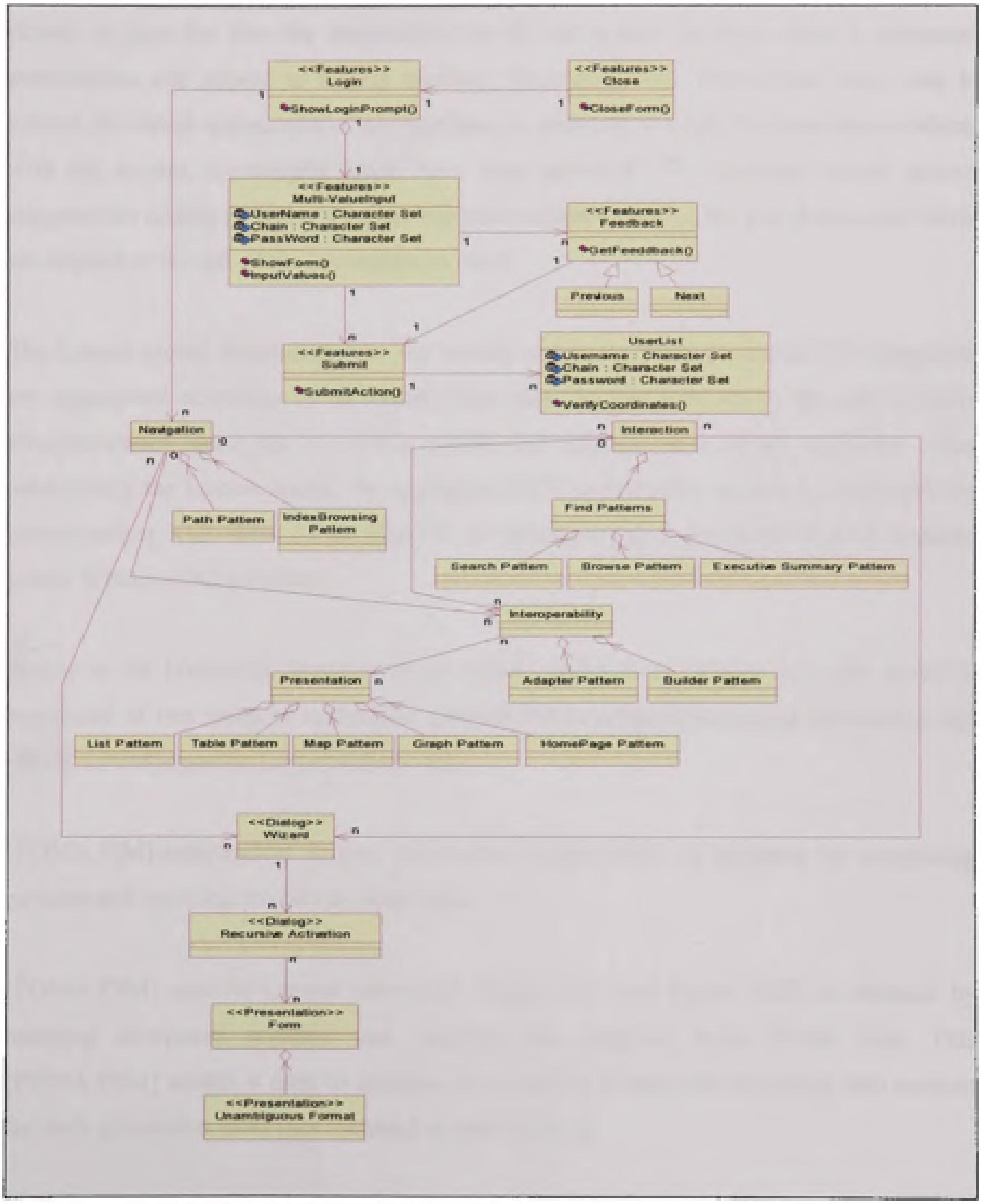

<<Features>> Login
ShowLoginPrompt()
<<Features>> Close
CloseForm()
<<Features>> Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
<<Features>> Feedback
GetFeedback()
Previous
Next
<<Features>> Submit
SubmitAction()
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Presentation
Adapter Pattern
Builder Pattern
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
<<Dialog>> Wizard
<<Dialog>> Recursive Activation
<<Presentation>> Form
<<Presentation>> Unambiguous Format

Figure 6.20 UML class diagram of the PSM Presentation model for a PDA platform.

In the Layout model, the style attributes, that have not yet been defined, are set in keeping with the standards set for the "Environmental Management Interactive System". According to the House Style pattern (which is applicable here), colours, fonts, and layouts should be chosen to give the user the impression that all the system windows share a consistent presentation and appear to belong together. Cascading style sheets have been used to control the visual appearance of the interface. In addition, to assist the user when working with the system, meaningful labels have been provided. The Labeling Layout pattern suggests the adding of labels for each interaction element. Using the grid format, the labels are aligned to the left of the interaction element.

The Layout model determines how the loosely connected XUL (Appendix III) fragments are aggregated according to an overall floor plan. In this case study, the task is fairly straightforward since the UI is not nested and consists of a single container. After establishing the Layout model, the aggregated XUL code can be rendered, along with the corresponding XUL skins, as the final UI. All interfaces are shown in the final UI rendered on the Windows XP platform.

Acting in the horizontal direction of the POMA architecture (Figure 3.2), this model is composed of two types of sub-model, [POMA.PIM]-independent Layout sub-model, and [POMA.PSM]-specific Layout sub-model.

[POMA.PIM]-independent Layout sub-model (Figure 6.21) is obtained by composing patterns and applying the composition rules.

[POMA.PSM]-specific Layout sub-model (Figure 6.22 and Figure 6.23) is obtained by mapping composed patterns and applying the mapping rules (Table 6.6). This [POMA.PSM] model is used to generate the system's source code by taking into account the code generation rules (not included in this research).

Figure 6.21 represents a UML class diagram of the PIM Layout model which is composed of several patterns. This model underwent mapping by applying the mapping rules (Table 6.6) to obtain another model, which is called the PSM Layout model (Figure 6.22 for a laptop platform and Figure 6.23 for a PDA platform).

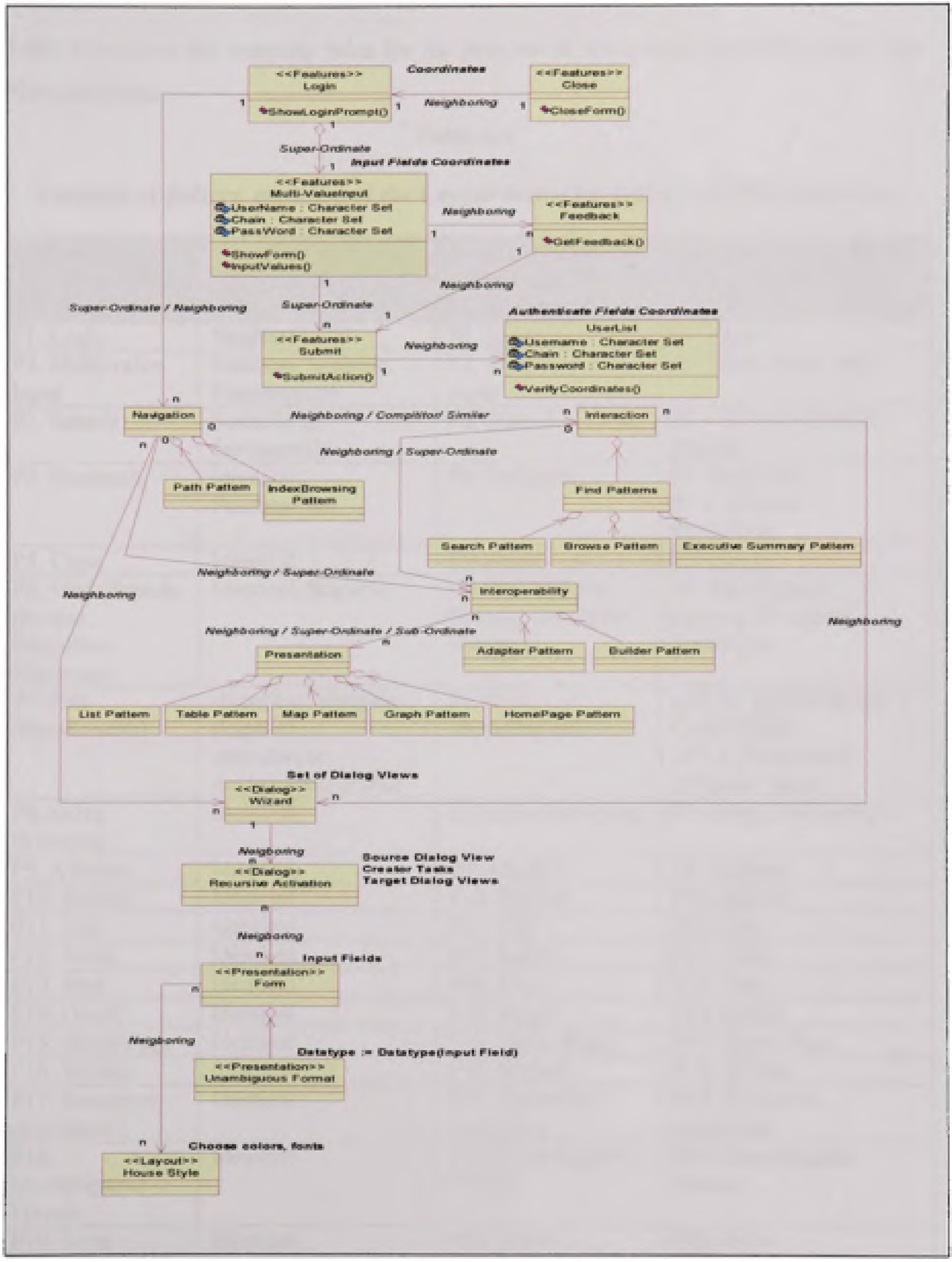

<<Features>>
Login
ShowLoginPrompt()
Coordinates
<<Features>>
Close
CloseForm()
Neighboring
Super-Ordinate
Input Fields Coordinates
<<Features>>
Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
Neighboring
<<Features>>
Feedback
GetFeedback()
Neighboring
Super-Ordinate / Neighboring
Super-Ordinate
Authenticate Fields Coordinates
<<Features>>
Submit
SubmitAction()
Neighboring
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
Navigation
Neighboring / Compititor/ Similar
Interaction
Neighboring / Super-Ordinate
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Neighboring / Super-Ordinate
Neighboring
Interoperability
Neighboring
Neighboring / Super-Ordinate / Sub-Ordinate
Presentation
Adapter Pattern
Builder Pattern
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
Set of Dialog Views
<<Dialog>>
Wizard
Neigboring
<<Dialog>>
Recursive Activation
Source Dialog View
Creator Tasks
Target Dialog Views
Neigboring
Input Fields
<<Presentation>>
Form
Neigboring
Datatype := Datatype(Input Field)
<<Presentation>>
Unambiguous Format
Choose colors, fonts
<<Layout>>
House Style
1
n
0

Figure 6.21 UML class diagram of a PIM Layout model.

Table 6.6 shows the mapping rules for the patterns of the Layout model for laptop and PDA platforms.

Table 6.6

Example of pattern mapping of the Layout model for laptop and PDA platforms

Patterns of Microsoft Platform	Type of Mapping	Replacement patterns for Laptop platform	Replacement patterns for PDA platform
P1. Login	Identical	P1. Login	P1. Login
P2. Multi-value Input	Identical, Scalable, Fundamental	P2. Multi-value Input	P2. Multi-value Input
P3. Submit	Scalable or fundamental	P3. Submit	P3.s. Submit (Smaller button)
P4. Feedback	Identical, Fundamental	P4. Feedback	P4. Feedback P4.1. Previous P4.2. Next
P5. Close	Identical	P5. Close	P5. Close
P6. Find (Search, Browse, Executive Summary)	Identical, Scalable	P6. Find (Search, Browse, Executive Summary	P6. Find (Search, Browse, Executive Summary)
P7. Path (Breadcrumb)	-Identical, Scalable (Laptop) -Scalable or fundamental (PDA)	P7. Path (Breadcrumb)	- P7.1s. Shorter Bread Crumb Trial - P7.2. Drop-down "History" menu
P8. Index Browsing	Identical	P8. Index Browsing	P8. Drop-down menu
P9. Adapter	Identical	P9. Adapter	P9. Adapter
P10. Builder	Identical	P10. Builder	P10. Builder
P11. List	Identical	P11. List	P11. List
P12. Table	Identical	P12. Table	P12. Table
P13. Map	Identical	P13. Map	P13. Map
P14. Graph	Identical	P14. Graph	P14. Graph
P15. Home Page	Identical	P15. Home Page	P15. Home Page
P16. Wizard	Identical	P16. Wizard	P16. Wizard
P17. Recursive Activation	Identical	P17. Recursive Activation	P17. Recursive Activation
P18. Unambiguous Format	Identical	P18. Unambiguous Format	P18. Unambiguous Format
P19. Form	Identical	P19. Form	P19. Form
P20. House Style	Identical	P20. House Style	P20. House Style

After the mapping, the PSM Layout model is obtained for a laptop platform – Figure 6.22.

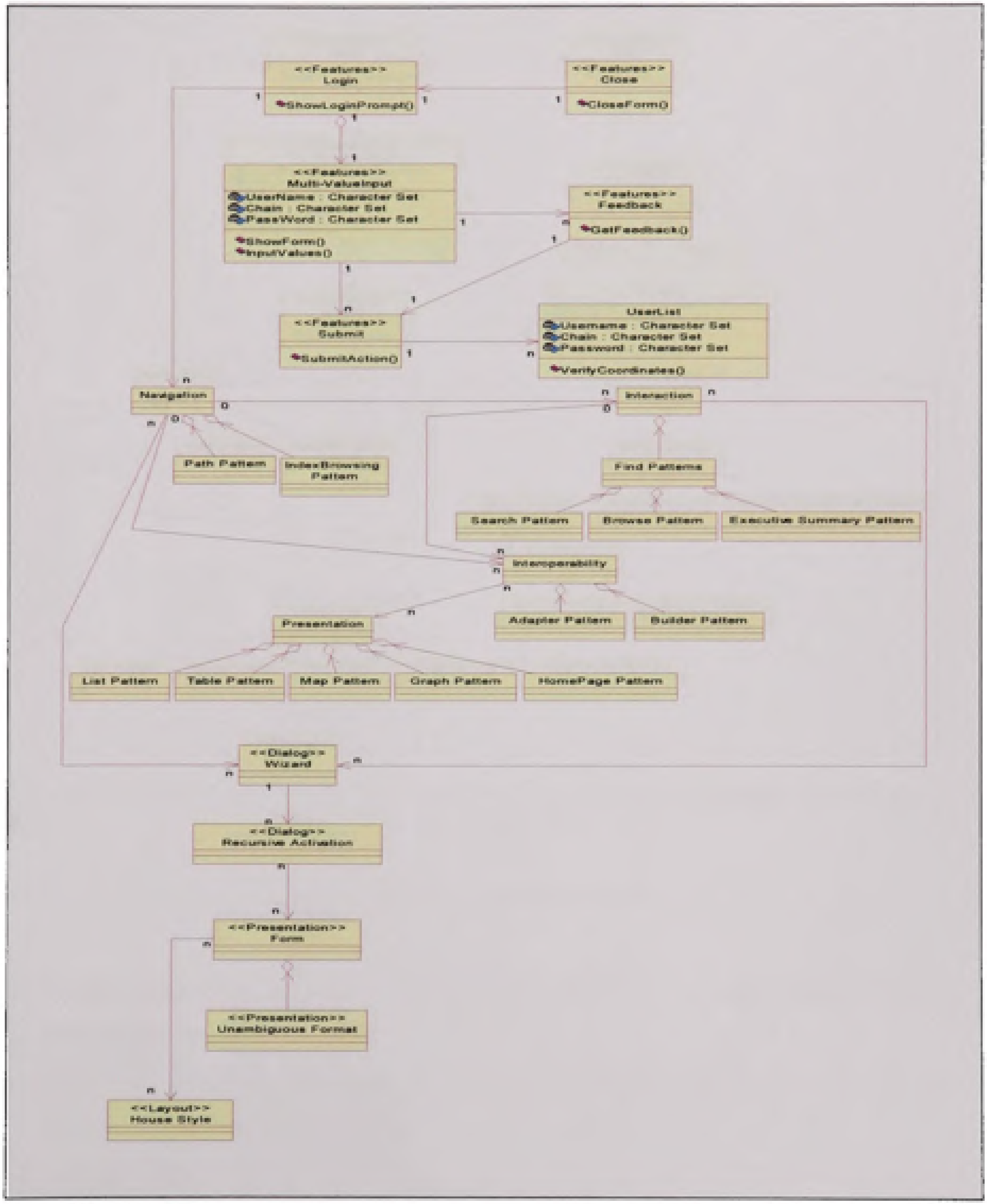

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
<<Features>>
Submit
SubmitAction()
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Adapter Pattern
Builder Pattern
Presentation
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
<<Dialog>>
Wizard
<<Dialog>>
Recursive Activation
<<Presentation>>
Form
<<Presentation>>
Unambiguous Format
<<Layout>>
House Style

Figure 6.22 UML diagram of PSM Layout Model for a Laptop platform.

After the mapping, the PSM Layout model is obtained for a PDA platform – Figure 6.23.

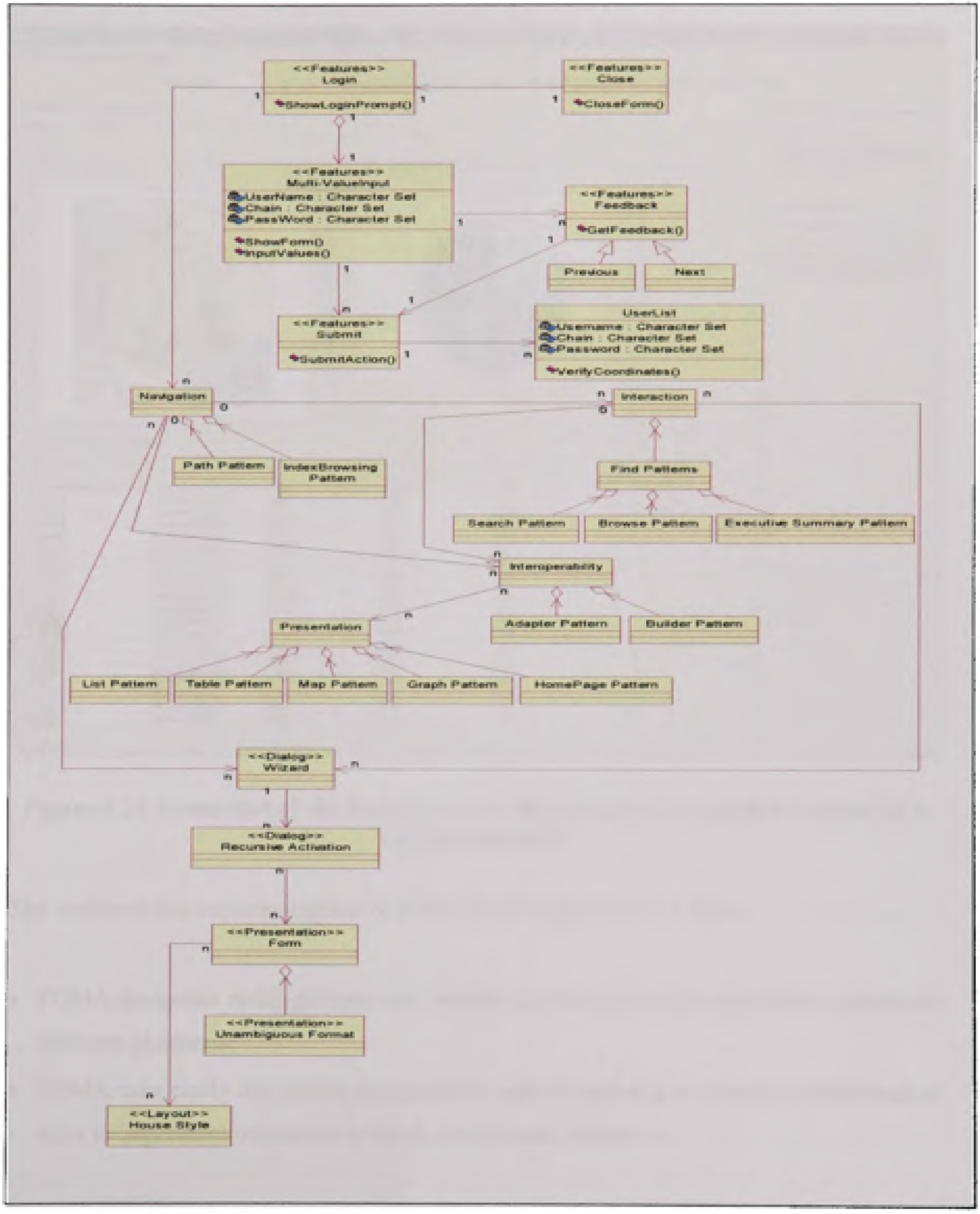

<<Features>>
Login
ShowLoginPrompt()
<<Features>>
Close
CloseForm()
<<Features>>
Multi-ValueInput
UserName : Character Set
Chain : Character Set
PassWord : Character Set
ShowForm()
InputValues()
<<Features>>
Feedback
GetFeedback()
Previous
Next
<<Features>>
Submit
SubmitAction()
UserList
Username : Character Set
Chain : Character Set
Password : Character Set
VerifyCoordinates()
Navigation
Interaction
Path Pattern
IndexBrowsing Pattern
Find Patterns
Search Pattern
Browse Pattern
Executive Summary Pattern
Interoperability
Adapter Pattern
Builder Pattern
Presentation
List Pattern
Table Pattern
Map Pattern
Graph Pattern
HomePage Pattern
<<Dialog>>
Wizard
<<Dialog>>
Recursive Activation
<<Presentation>>
Form
<<Presentation>>
Unambiguous Format
<<Layout>>
House Style
1
n
0

Figure 6.23 UML diagram of PSM Layout Model for a PDA platform.

Figure 6.24 is the final layout of the "Environmental Management Interactive System".

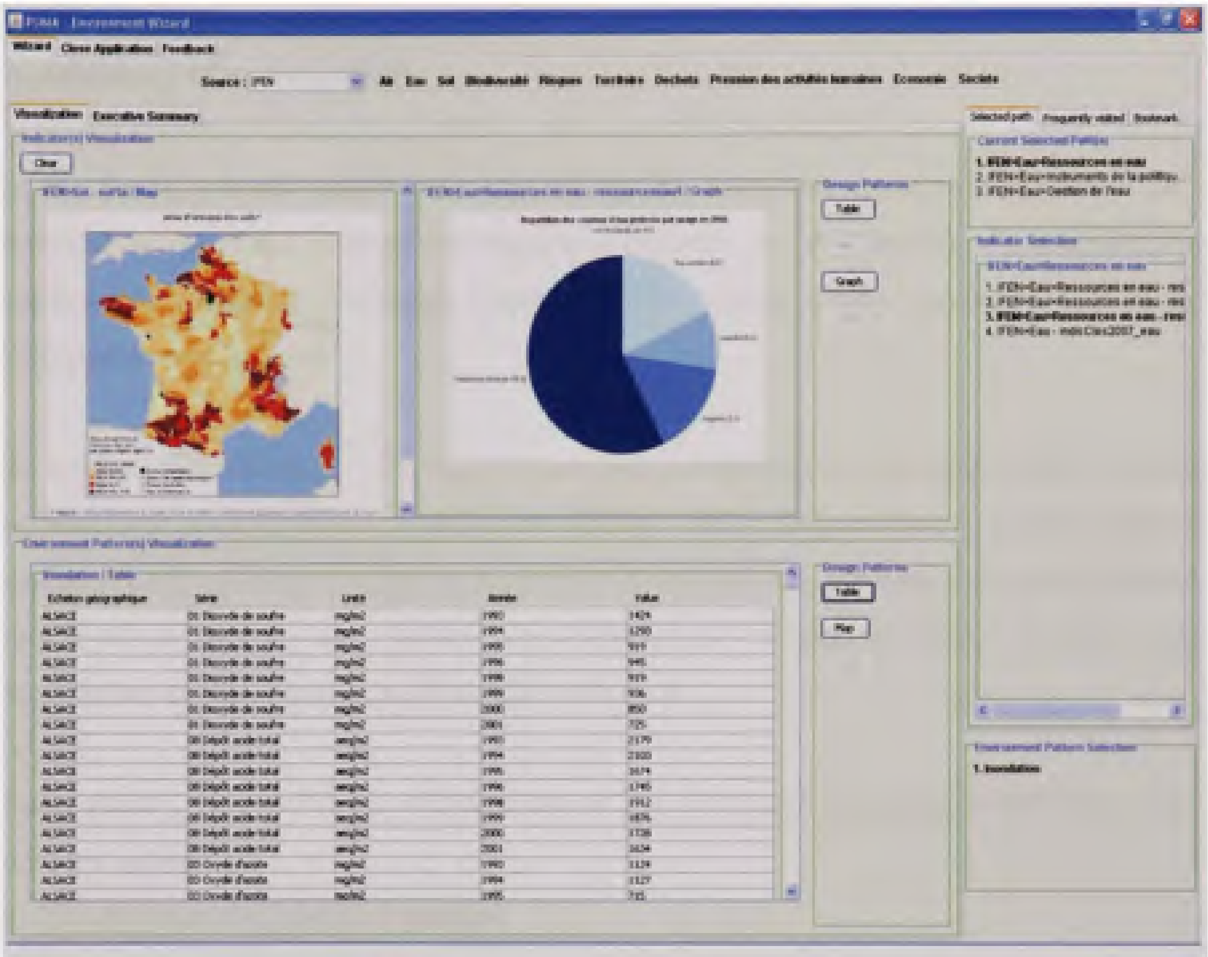

POMA - Environment Wizard
Wizard
Close Application
Feedback
Source :
Visualization
Executive Summary
Clear
Design Patterns
Table
Graph
Map
Selected path
Frequently visited
Bookmark

Figure 6.24 Screenshot of the Environmental Management Interactive System for a Laptop platform.

The results of this experimentation of POMA architecture are as follows:

- POMA integrates easily patterns and models together to design interactive systems for different platforms;
- POMA uses easily the pattern composition, pattern mapping and model transformation rules to implement interactive systems for different platforms.

CONCLUSION

A. Summary of investigations

In this thesis, a novel architecture is introduced for interactive systems engineering. This architecture combines patterns and models to facilitate the engineering of multi-platform applications including traditional computer, PDA, mobile phone, interactive television, laptop and palmtop.

To achieve this objective, a literature review was conducted on existing interactive system architectures such as N-tier architectures (MVC, J2EE, and Zackman), pattern-oriented design (POD), pattern-supported approach (PSA) and architecture based on models such as model-driven architecture (MDA). The foundations, shapes or forms, strengths and weaknesses of existing architectures were reviewed.

Subsequently, the different components of the proposed architecture (POMA) were detailed including:

1. The different categories of patterns;
2. The PIM and PSM models;
3. The pattern composition rules to select and compose patterns corresponding to each type of PIM model;
4. The pattern mapping rules to map the patterns and PIM models to produce PSM models for multiple platforms;
5. The transformation rules for transforming PIM to PIM models and PSM to PSM models.

A prototype of this case study was developed for an environmental management interactive system. This prototype was developed in Java Eclipse tool. In this case study, patterns were identified and applied for each of the models that were used during development.

The main purpose of the prototype of this case study is to show that model-driven architecture development consists of model transformation and that mapping rules from the abstract to the concrete models are specified and – more importantly – automatically supported by tools.

In the case study, UML notation was used to design the five models (Domain, Task, Dialog, Presentation and Layout). XML notation was also used to describe the five models and the different types of patterns proposed by the POMA architecture. UML and XML allow one to communicate the modeling semantics between the different models, helping tailor the application and corresponding models to different platform and user roles.

B. Key Contributions

This research created a practical multi-platform architecture for interactive systems engineering. The main contributions are:

1. The creation of six architectural levels and categories of patterns (Navigation patterns, Interaction patterns, Visualization patterns, Presentation patterns, Interoperability patterns, and Information patterns) (Taleb et al., 2006), (Taleb et al., 2007a) and (Taleb et al., 2007c);
2. The creation of different relationships between patterns which are used to create a pattern–oriented design using composition rules and mapping rules and to generate specific implementations suitable for different platforms from the same pattern-oriented design (Taleb et al., 2006) and (Taleb et al., 2007c);
3. The use of Five categories of models: Domain model, Task model, Dialog model, Presentation model and Layout model (Taleb et al., 2007b) and (Taleb et al., 2008c);
4. The creation of different model transformation rules to transform only the PIM and PSM models between them such as: PIM to PIM, PIM to PSM, and PSM to PSM (Taleb et al., 2008c);

5. Development of the "Environmental Management Interactive System" case study. The case study illustrates and clarifies the core ideas of this research approach and its applicability and relevance to multi-platform development (Taleb et al., 2008d).

Various contributions documented in this thesis have been published at conferences and in journals. The list follows:

Conference Papers

Published

1. M. Taleb, H. Javahery, A. Seffah, 2006, 'Pattern-Oriented Design Composition and Mapping for Cross-Platform Web Applications', the XIII International Workshop, DSVIS 2006, July 26-28 2006, Trinity College Dublin Ireland, DOI 10.1007/978-3-540-69554-7, ISBN 978-3-540-69553-0, Vol. 4323/2007, Publisher Springer-Verlag Berlin Heidelberg, Germany.
2. M. Taleb, A. Seffah, A. Abran, 2007, 'Pattern-Oriented Architecture for Web Applications', 3rd International Conference on Web Information Systems and Technologies (WEBIST), March 3-6, 2007, ISBN 978-972-8865-78-8, pp. 117-121, Barcelona, Spain.
3. M. Taleb, A. Seffah, and A. Abran, 2007, 'Model-Driven Design Architecture for Web Applications', The 12th International Conference on Human Centered Interaction International (FIC-HCII), July 22-27, 2007, Beijing International Convention Center, Beijing, P.R. China, Vol. 4550/2007, pages 1198-1205, Publisher Springer-Verlag Berlin Heidelberg, Germany
4. M. Taleb, A. Seffah, and A. Abran, 2007, 'Patterns-Oriented Design for Cross-Platform Web-based Information Systems', The 2007 IEEE International Conference on Information Reuse and Integration (IEEE IRI-07), August 13-15, 2007, pages 122-127, Las Vegas, USA.

Submitted

1. M. Taleb, A. Seffah and A. Abran. 2008b. Patterns + Personas = A Human-Centric Infrastructure for Web Applications Design. 9th International Conference on Web Engineering (ICWE), June 24-26, 2009, San Sebastian, Spain.

Journal Papers

Accepted

1. M. Taleb, A. Seffah and A. Abran, 2008, 'Reconciling Usability and Interactive System Architecture Using Patterns', Journal of Systems and Software, to be published in 2008. (Accepted on April 10th, 2008).

Submitted

1. M. Taleb, A. Seffah and A. Abran, 2008, "Investigating Model-Driven Architecture for Web-based Interactive Systems", Journal of eMinds.
2. M. Taleb, A. Seffah and A. Abran, 2008, 'POMA: A Pattern-Oriented and Model-Driven Architecture', Journal of Software - Practice and Experience.
3. A. Seffah and M. Taleb, 2009, 'Tracing the Evolution of Patterns as a Design Tool', Journal of Innovations in Systems and Software Engineering.

Chapters in a book

1. H. Javahery, A. Deichman, A. Seffah, and M. Taleb, 2007, '*A User-Centered Framework for deriving a conceptual design from user experiences. Leveraging personas and patterns to create usable designs*', In A. Seffah, J. Gulliksen, and M. Desmarais, (eds), Human-Centered Software Engineering, Volume II. Software Engineering Models, Patterns and Architectures for HCI, Chapter 4, May 28th 2007, Wiley, New York, USA.
2. M. Taleb, A. Seffah, and D. Engelberg, 2007, '*From User Interface Usability to the Overall Usability of Interactive Systems: Adding Usability in System Architecture*', In A. Seffah, J. Gulliksen, and M. Desmarais, eds, Human-Centered Software Engineering, Volume II, Software Engineering Models, Patterns and Architectures for HCI, Chapter 9, May 28th 2007, Wiley, New York, USA.

In addition to these main contributions, this research has developed an architecture that facilitates usability (Taleb et al., 2007d).

C. Implications for software engineering theory

This architecture opens a new research avenue to the use of models and patterns together in the design process. Compared to existing architectures, this research introduced:

1. Novel architecture called Pattern-Oriented and Model-driven Architecture (POMA) for interactive systems (section 3.2);
2. Novel pattern selection and composition rules (section 4.1.2);
3. Novel pattern mapping rules (section 4.1.3);
4. Novel model transformation rules (section 5.2).

D. Practical implications

The results of this research have practical implications for interactive systems engineering. The proposed POMA architecture allows the industry to improve and to facilitate the development of interactive systems using patterns and models in order to obtain the user interfaces that are more convivial and easy to use.

The application of different types of patterns provides a better understanding of interrelated and viewed data on different screens and to structure information for a better visualization. The use of various models offers the industry a flexibility framework to understand design problems raised by research and / or by the industry itself.

Moreover, the results of this research can increase mutual understanding between software engineers and HCI experts to address the problem mentioned by (Jerome and Kazman, 2007) and (Donyaee, 2008):

> "Software engineers and HCI practitioners tend to interact with each other late in the software life cycle" (Jerome and Kazman, 2007) and (Donyaee, 2008).

E. Limitations and strengths

The Model-View-Controller (MVC), Model-View-Presenter (MVP), Presentation-Abstraction-Control (PAC), Seeheim, Arch/Slinky, PAC-Amadeus, and POMA architectures are similar in a number of ways, but each has evolved to address a slightly different concern. By becoming familiar with these architectures and other related architecture models, developers and architects will be better equipped to choose appropriate solutions in their design endeavors, or possibly in the creation of future pattern-oriented and model-driven architectures.

Current limitations of POMA include the following:

- There is a need to define measures to assess objectively the applicability of the patterns that could be used in POMA;
- The patterns do not provide sufficient provisions for dealing with the platform-independent specification of interfaces, the platform-specific form of those interfaces, nor the eventual implementation of those interfaces;
- POMA does not encourage the designer to consider other aspects of the dialog which are very important to the user (help function or error-handling);
- POMA does not facilitate the use of design constraints or descriptions of the interface which are of great importance to the designer (Booch et al., 1999), (Myers, 1986), (Myers and Buxton, 1986) and Myers, 1990);
- Patterns show weakness signs in their programming languages;
- Finding and applying the appropriate architectural patterns in practice still remains largely an ad hoc and unsystematic process, e.g. there is a lack of consensus in the community with respect to the "philosophy" and granularity of architectural patterns and a lack of a mature pattern language;
- The need for affordable educational opportunities that focus on the style guidelines and patterns for both the design and use of Interface Specification Meta-Language (ISMLs);
- Limited use of the ISO/IEC 9126-1 (2001) Standard and other quality attributes, such as communicability, learnability, maintainability, and usability.

Further research is required to address these limitations, one by one.

The strengths of POMA architecture include the following:

- POMA facilitates the use of patterns by beginners as well as experts;
- POMA supports the automation of both the pattern-driven and model-driven approaches to design;
- POMA supports the communication and reuse of individual expertise regarding good design practices;
- POMA can integrate all the various new technologies including, but not limited to, traditional office desktops, laptops, palmtops, PDAs with or without keyboards, mobile telephones, and interactive televisions.

F. Further research

Among the next steps required to develop POMA are:

- Standardization of POMA architecture to all types of systems, not only to multi-platform interactive systems;
- Description of a process for the generation of a source code from the five POMA PSM models;
- Development of a tool that automates the POMA architecture-based engineering process;
- Quality Assurance of the applications produced, since a pattern-oriented architecture will also have to permit the encapsulation of quality attributes and to facilitate prediction;
- Validation of the migration, the usability and overall quality of POMA architecture for interactive systems using different existing methods ;
- Evaluation of the effectiveness and learning time of POMA architecture for novices and experts users.

APPENDIX I

GLOSSARY OF TERMS

API Server	Derived from BGI, the NSAPI, ISAPI, WSAPI and other API servers are functional equivalents of BGI and generally incompatible.
Architecture	1. The *architecture* of a system is a specification of the parts and connectors of that system and the rules for the interactions of the parts using connectors (Shaw and Garlan, 1996). 2. "The software *architecture* of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components, and the relationships among them" (Bass et al., 2003). Model-Driven Architecture prescribes certain kinds of models to be used, how those models may be prepared and the relationships between the different kinds of models.
BGI	*BGI* is an interface used by the server to communicate with internal applications. This gateway to communication exchanges information with functions belonging to a library of dynamic links (.dll).
CGI	*CGI* is an interface used by the server to communicate with external applications. This gateway to communication exchanges information with scripts, programs and software.
Component	A *component* is a set of the objects combined together to form a unit.
Computation Independent Model (CIM)	A *computation independent model* is a view of a system from the computation independent viewpoint. A CIM does not show details of the structure of systems. A CIM is sometimes called a domain model; a vocabulary that is familiar to the practitioners of the domain in question is used in its specification.
Computation Independent Viewpoint (CIV)	The *computation independent viewpoint* focuses on the environment of the system, and the requirements for the system; the details of the structure and processing of the system are hidden or as yet undetermined.

CORBA	*CORBA* is a standard for architectures of the distributed objects intended for small to large applications.
HTML	Static language of programming of documents on the Internet describing the structure and not the appearance of a document.
Implementation	An *implementation* is a specification which provides all the information needed to construct a system and to put it into operation.
Java	*Java* is the most powerful language of the Internet. It is used for the Client/Server applications of average to great scale, as much for the user interface as for the server.
JDBC	*JDBC* is a programming interface which allows communication between Java programs, more particularly the user interfaces of the client, with a database.
Middleware	A *middleware* is software, which supports communication between the tier components of an interactive system, two or several interactive systems and interactive systems and shared services.
Model	1. A *model* of a system is a description or specification of that system and its environment for a specific purpose, which may be represented graphically or textually (Si Alhir, 2003). 2. In MDA, a *model* is a representation of a part of the function, structure and/or behavior of a system.
Model Transformation	*Model transformation* is the process of converting one model to another model of the same system.
Model-Driven	1. In MDA, a *model-driven* provides a means for using models to direct the course of understanding, for design, construction, deployment, operation, maintenance and modification. 2. A *model-driven* approach focuses on models to work with systems, including: understanding, designing, constructing, deploying, operating, maintaining, and modifying them (Si Alhir, 2003).
Model-Driven Architecture (MDA)	1. *MDA* is a software design architecture. 2. An approach to IT system specification that separates the specification of functionality from the specification of the implementation of that functionality on a specific technology platform.

Pattern Mappings	1. *Pattern mapping* is the process of creating a design of specific models for each platform called platform-specific model (PSM) – from PIM and mapping rules. 2. A *mapping* is a specification (or transformation specification), including rules and other information, for transforming a PIM model to produce PSM for a specific platform (Si Alhir, 2003).
Pervasive Services	*Pervasive services* are services available in a wide range of platforms.
Platform	A *platform* is a set of subsystems and technologies that provide coherent sets of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.
Platform Independent	*Platform independence* is a quality that a model may exhibit, independent of the features of a platform of any particular type.
Platform Independent Model (PIM)	A *platform independent model* is a view of a system from the platform independent viewpoint. A PIM exhibits a specified degree of platform independence to be suitable for use with a number of different platforms of similar type.
Platform Independent Viewpoint (PIV)	The *platform independent viewpoint* focuses on the operation of a system while hiding the details necessary for a particular platform. A platform independent view shows that part of the complete specification that does not change from one platform to another. A platform independent view may use a general purpose modeling language or a language specific to the area in which the system will be used.
Platform Model	A *platform model* provides a set of technical concepts representing the different parts that make up a platform and the services provided by that platform and provides, for use in a platform specific model, concepts representing the different elements to be used in specifying the use of the platform by an application.
Platform Specific Model (PSM)	A *platform specific model* is a view of a system from the platform specific viewpoint. A PSM combines the specifications in the PIM with the details that specify how that system uses a particular type of platform.

Platform Specific Viewpoint (PSV)	The *platform specific viewpoint* combines the platform independent viewpoint with an additional focus on the detail of the use of a specific platform by a system.
RMI	*RMI* is API allowing the invocation of methods on distributed objects intended for the applications of small and average size.
Servlet	*Servlet* is an alternative to the .dll used for the BGI. Servlets are written in Java and are generally portable from one platform to another. The server must however support them.
System	1. A set of assembled elements, which interact in a manner consistent or predictable in an environment and in pre-defined or observed conditions. 2. MDA concepts are presented here in terms of existing or planned systems. That *system* may include anything: a program, a single computer system, a combination of parts of different systems, a federation of systems, each under separate control. 3. A *system* (or physical system) is a collection of elements organized together for a specific purpose (Si Alhir, 2003).
View	A *view* or viewpoint model of a system is a representation of that system from the perspective of a chosen viewpoint (IEEE, 2000).
Viewpoint	A *viewpoint* on a system is a technique for abstraction using a selected set of architectural concepts and structuring rules, in order to focus on particular concerns within that system. Here 'abstraction' is used as the process of suppressing selected details to establish a simplified model. The concepts and rules may be considered to form a viewpoint language.
Web Application	A *web application* is a computer program that users invoke by using a web browser to contact a web server via the Internet. Users and browsers are typically unaware of the difference between contacting a web server which fronts for a statically built website and a web server which fronts for a web application. But unlike a static website, a *web application* creates its "pages" dynamically. A website that is dynamically constructed uses a computer program to provide the dynamism. These types of dynamic applications can be written in any number of computer languages.

Workflow	A *workflow* is coordinated set of actions or operations which are related, in series or in parallel, in order to achieve a common goal: • Actions are the activities executed by humans; • Operations are the activities executed and controlled automatically by a system management process.

APPENDIX II

TECHNICAL REPORT OF INTERACTIVE SYSTEM DEVELOPMENT TOOLS: TRENDS AND CHALLENGES IN INTERACTIVE SYSTEM DEVELOPMENT TOOLS: REQUIREMENTS FOR PATTERN-ORIENTED AND MODEL-BASED ARCHITECTURE

1. Introduction

In Software Engineering, "Interactive System" is a system accessed by interfaces over a network such as the Internet, intranet, extranet or a by traditional medium (Wikipedia).

Interactive systems are popular due to the ubiquity of the browser as a client, sometimes called a thin client. The ability to update and maintain interactive systems without distributing and installing systems on potentially thousands of client computers is a key reason for their popularity. Interactive systems are used to implement for example ; Webmail, online retail sales, online auctions, wikis, discussion boards, Weblogs, MMORPGs and many other functions (Wikipedia).

In earlier types of client-server computing, each system had its own client program which served as its user interface and had to be separately installed on each user's personal computer. An upgrade to the server part of the system would typically require an upgrade to each user workstation, adding to the support cost and to decreasing productivity (Wikipedia).

In contrast, interactive systems dynamically generate a series of documents in a standard format supported by common interfaces such as HTML/XHTML. Client-side scripting in a standard language such as JavaScript is commonly included to add dynamic elements to the user interface. Generally, each individual interface is delivered to the client as a static document, but the sequence of pages can provide an experience, as user input is returned through form elements embedded in the interface markup. During the session, the interactive system interprets and displays the interfaces and acts as the *universal* client for any system (Wikipedia).

The interface places very few limits on client functionality. Through Java, JavaScript, Flash and other technologies, system-specific methods such as drawing on the screen, playing audio, and access to the keyboard and mouse are all possible. General-purpose techniques such as drag and drop are also supported by Java, though this may be simpler with current JavaScript libraries. Developers often use client-side scripting to add functionality, especially to create an interactive experience that does not require page reloading (which many users find disruptive). Recently, technologies have been developed to coordinate client-side scripting with server-side technologies such as PHP. Ajax, a development technique using a combination of various technologies, is an example of a technology which creates a more interactive experience (Wikipedia).

A significant advantage in building interactive systems to support standard browser features is that they should perform as specified regardless of the operating system or OS version installed on a given computer. Rather than creating clients for MS Windows, Mac OS X, GNU/Linux, and other operating systems, this system can be written once and deployed almost anywhere. However, inconsistent implementations of the HTML, CSS, DOM and other browser specifications can cause problems in interactive system development and support. Additionally, the ability of users to customize many of the display settings of their browser (such as selecting different font sizes, colors, and typefaces, or disabling scripting support) can interfere with consistent implementation of an interactive system (Wikipedia).

Another (less common) approach is to use Macromedia Flash or Java applets to provide some or all of the user interface. Since most, for example, Web browsers include support for these technologies (usually through plug-ins), Flash- or Java-based systems can be implemented with much of the same ease of deployment. Because they allow the programmer greater control over the interface, these systems bypass many browserconfiguration issues, although incompabilities between Java or Flash implementations on the computer can introduce different complications. Because of their architectural similarities to traditional client-server systems, with a somewhat "thick" client, there is some dispute over whether to call systems of this sort "interactive systems" (Wikipedia).

Though many variations are possible, an interactive system is commonly structured as a three-tiered system. In its most common form, an interface is the first tier, an engine using some dynamic content technology (e.g., CGI, PHP, Java Servlets or Active Server Pages (ASP)) is the middle tier, and a database is the third tier. The interface sends requests to the middle tier, which services them by making queries and updates against the database to generate a user interface (Wikipedia).

Interfaces have increasingly replaced what have previously been thought of as traditional, single-user systems. For example, Microsoft HTML Help replaced Windows Help as the primary help system in Microsoft Windows. Like their networked brethren, such systems generate web documents as their user interface and send them (sometimes via an embedded HTTP server) to a local Web browser component, which then renders the pages for the user and returns user input to the system. Interactive systems powered by embedded servers have also become commonplace as user interfaces for configuring network components such as servers, switches, routers, and gateways (Wikipedia).

An emerging strategy for application system companies is to provide access to systems previously distributed as local systems. Depending on the type of system, the development of an entirely different interface may be required, or merely adapting an existing system to use a different presentation technology. These programs allow the user to pay a monthly or yearly fee for use of a software system without having to install it on a local hard drive. A company which follows this strategy is known as a system service provider (ASP). ASPs are currently attracting a great deal of attention in the software industry (Wikipedia).

While many interactive systems are written directly in PHP or mod_perl, there are many interactive system architectures which automate the process by allowing the programmer to define a higher level description of the program. In addition, there is potential for the development of systems on Internet Operating Systems, although there are not many viable platforms that currently fit this model (Wikipedia).

The use of interactive system architectures can often reduce the number of errors in a program, both by making the code more simple, and by allowing one team to concentrate uniquely on the architecture. In systems which are exposed to constant hacking attempts via the Internet, security-related problems caused by errors in the program are a main issue (Wikipedia).

As of 2006, Java remains one of the most common programming languages for writing interactive systems. This is especially true for enterprise systems (usually referred to as enterprise interactive systems). J2EE (a Java programming platform) provides several useful components (JavaServer Pages, servlets, client-side applets, Enterprise Java Beans, JDBC and several service technologies) for writing enterprise interactive systems. As of 2006, J2EE remains the standard in this area (Wikipedia).

The Web Application Security Consortium (WASC), CGI Security, and OWASP are projects developed with the intention of documenting how to avoid security problems in interactive systems (Wikipedia).

This appendix specifies and describes the whole of existing tools for interactive system development and gives the attributes of a good description of software. The principal objectives of this appendix are (Wikipedia):

- Clarify the different existing tools for interactive system development;
- Distinguish between the tools which are oriented pattern and others;

- Describe the different existing formalisms and notations suggested by specialists in the field for patterns, architectures and models.

The first section describes the system generator tools such as Content Management System (CMS). The second section will identify the different tools for Model-based Approach and Patterns. The last section presents the existing formalisms and notations to support the specification of patterns, architectures and models (Wikipedia).

2. Content Management System (CMS)

A **Content Management System** is a computer software system used for organizing and facilitating collaborative creation of documents and other content. A content management system is often an interactive system used for creating and managing interfaces and their contents. Alternatively, content management systems can also be used for storing and publishing documentation such as operating manuals, technical manuals and sales guides. There are many open-source and proprietary CMS solutions available, which is in fact true for most systems of any kind. The market for content management systems is quite fragmented (Wikipedia).

A content management system is essentially a way of separating visual presentation from actual content, whether that content includes photos, text or product catalogs. This separation allows one to accomplish several key tasks, including:

- **Automated Templating:** Create standard visual templates that can be automatically applied to new and existing content, creating one central place to change caracteristics across the content on a site;
- **Easily Editable Content:** Once the content is separate from the visual presentation of a site, editing usually becomes much easier and quicker to manipulate. Most CMS software includes WYSIWYG editing tools allowing non-technically trained individuals to easily create and edit content;

- **Scalable Feature Sets:** Most CMS have plug-ins or modules that can be easily installed to extend on existing site's functionality. For example, if one wanted to add a product catalog or chat functionally on a website, one could easily install a module/plug-in to add that functionality rather than hiring a developer to hard code that new functionality;
- **Web Standards Upgrades:** Active CMS solutions usually receive regular updates that include new feature sets to keep the system up to current web standards. These updates are usually designed for easy installation over/on-top of an existing website;
- **Community Support:** Most active CMS solutions have developer support forums. Since CMS users/developers are beginning from a common base, developers are more likely to encounter the same development challenges and can address those challenges as a community;
- **Lower Cost Maintenance:** CMS hosted sites are often easier and cheaper to maintain. Since any CMS powered website would have a community familiar with the tools of that specific CMS, it would be quite easy for a new developer to make updates or do maintenance;
- **Workflow management**: Workflow is the process of creating cycles of sequential and parallel tasks that must be accomplished in a CMS. For example, a user posts a story but it is not published on the website until the editor approves it.

An interactive system content management system runs on the system's server. Most systems provide controlled access for various ranks of users such as administrators, editors and ordinary content creators.

The content and all other information related to the site are usually stored in a server-based database system such as MySQL. The pages created by the content management system can be viewed by visitors to the site. Internally, many interactive system content management systems are written in the PHP programming language.

The following terms are often used in relation to interactive system content management systems but they may be neither standard nor universal:

- **Block** – A block is a link to a section of an interactive system. Blocks can usually be specified to appear on all interfaces of the system (for example in a left-hand navigation panel) or only on the home page;
- **Module** – A content module is a section of the interactive system, for example a collection of news articles or an FAQ section. Some content management systems may also have other special types of modules such as administration and system modules;
- **Theme** – A theme specifies the cosmetic appearance of every interface of an interactive system, controlling properties such as the colors and the fonts.

2.1 Example of tools for CMS

2.1.1 Zope: Tools available for accessing Zope

- Zope (Zope, 2008) is an architecture that allows developers of varying skill levels to build interactive systems;
- Zope is an open source tool for sophisticated interactive systems development;
- Zope allows for a combination of objects and creates a template (example: payment interface, reservation interface);
- Zope separates the content from presentation and interaction;
- All objects or components for user interfaces are the high level patterns;
- Zope resolves some problems in a Java development environment (example Eclipse tool);
- The user can create new objects and add them to the Zope environment.

Zope tool can generate:

- An interactive system;
- A portal;
- Different interfaces of an interactive system (www.welie.com or Design Patterns);
- Different types of interactive systems as the objects;
- A user's own system directly (e-commerce);
- Support for a Pattern-Oriented Approach;
- An extension to the HTTP Server.

Limitations of Zope tools according to the article (Grundy and Zou, 2004)

- Adaptation is difficult to achieve from one platform to another;
- There are rendering device problems. Such tools work reasonably well but do not support user and task adaptation well and require complex transformation scripts that have limited ability to produce good user interfaces across all possible rendering devices.

2.1.2 PhPNuk

PHP-Nuke (PhPNuk, 2006) is an interactive system-based automated news publishing and content management system (a 'nuke') based on PHP and MySQL. The system is fully controlled by a user interface. PHP-Nuke was originally a fork of the Thatware news portal system.

The main purpose of PHP-Nuke is to allow a developer to create a community-based portal (similar to that used by Slashdot) with an automated interactive system that allows users and editors to post news items (user-submitted news items are selected by editors). Users can comment on these articles using the "comments system".

Modules may be added to the PHP-Nuke system, allowing the developer to add more features (such as an Internet forum or calendar) to their PHP-Nuke installation in addition to the core modules such as News, FAQ and Private Messaging. The whole system is maintained by administrators using the web-based "admin section".

PHP-Nuke is able to support many languages, including English, French, Portuguese and Thai. Its look and feel can also be customized (to an extent), using the *Themes* system, although some people have found it difficult to make the site look any different to the standard column layout (as used by the program's official website).

PHP-Nuke has, in the past, been criticized for containing many security holes. SQL injection is one of the most widely-known flaws in PHP-Nuke's security, although other methods of gaining access to the admin panel of a site running PHP-Nuke have been found. In February, 2005, the Webmail module of PHP-Nuke was removed from all versions, due to security problems, at the request of EV1Servers.net, phpnuke.org's web hosting service. Since it contains a port of phpBB2, it also inherits phpBB's security flaws (Wikipedia).

2.2 List of Tools in a Content Management Systems

The following table is a list of notable **Content Management Systems (CMS)** that are used to organize and facilitate collaborative content creation. Many of CMS are built on top of separate content management frameworks (Wikipedia).

Free and open source software

Other interactive system development tools are also represented on the table below.

CMS Tools	Platform	Content Creation	Content Management	Access Control	Other Requirements
DreamWeaver	Windows	Yes	Yes	No	Web Drive
Web Drive	Windows	Yes	Yes	No	
MS Office	Windows	Yes	Yes	No	Web Drive
FrontPage	Windows	No	No	No	
External Editor	Windows / Unix / Linux	No	Yes	No	External Editor Client File editor(s) of your choosing (VI, Notepad, PhotoShop, etc.)
Cadaver	Unix / Linux / MacOS X	Yes	Yes	No	Console Editor of Choosing
Goliath	MacOS X	Yes	Yes	No	
Zope Management Interface	All platforms	Yes	Yes	Yes	Web Browser
PhPNuk	All platforms	Yes	Yes	Yes	Web Browser
Aegir (previously Aegir CMS)	Midgard add-on	Yes	Yes	Yes	Web Browser
Alfresco	Java	Yes	Yes	Yes	Web Browser

Apache Lenya	Java, XML, built on top of Apache Cocoon	Yes	Yes	Yes	Web Browser
Ariadne		Yes	Yes	Yes	Web Browser
b2evolution	PHP	Yes	Yes	Yes	Web Browser
Bblog	PHP + Smarty	Yes	Yes	Yes	Web Browser
Blockstar	Java	Yes	Yes	Yes	Web Browser
BLOG:CMS	PHP	Yes	Yes	Yes	Web Browser
blosxom	Perl	Yes	Yes	Yes	Web Browser
Bricolage	Perl on mod perl	Yes	Yes	Yes	Web Browser
Caravel CMS	PHP	Yes	Yes	Yes	Web Browser
Chlorine Boards	PHP	Yes	Yes	Yes	Web Browser
CivicSpace	PHP	Yes	Yes	Yes	Web Browser
CMScout	PHP	Yes	Yes	Yes	Web Browser
CMSimple	PHP	Yes	Yes	Yes	Web Browser
Community Server	ASP.NET	Yes	Yes	Yes	Web Browser
Daisy (CMS)	Java, XML, built on top of Apache Cocoon	Yes	Yes	Yes	Web Browser
DBHcms	PHP	Yes	Yes	Yes	Web Browser
DotNetNuke	VB.NET	Yes	Yes	Yes	Web Browser
DragonflyCMS	PHP	Yes	Yes	Yes	Web Browser
Drupal	PHP4-5	Yes	Yes	Yes	Web Browser
e107	PHP	Yes	Yes	Yes	Web Browser
eGroupWare	PHP	Yes	Yes	Yes	Web Browser
Epiware	PHP	Yes	Yes	Yes	Web Browser
eZ publish	PHP	Yes	Yes	Yes	Web Browser
Fedora	Java	Yes	Yes	Yes	Web Browser
Geeklog	PHP	Yes	Yes	Yes	Web Browser
Jahia	Java on Windows NT, Linux, or Solaris	Yes	Yes	Yes	Web Browser

jAPS – java Agile Portal System	Java, XML on Windows or Linux	Yes	Yes	Yes	Web Browser
Joomla!	PHP	Yes	Yes	Yes	Web Browser
Kwiki	Perl	Yes	Yes	Yes	Web Browser
Lyceum	PHP	Yes	Yes	Yes	Web Browser
Magnolia	Java	Yes	Yes	Yes	Web Browser
Mambo	PHP	Yes	Yes	Yes	Web Browser
MediaWiki	PHP	Yes	Yes	Yes	Web Browser
Midgard CMS	PHP (Midgard framework)	Yes	Yes	Yes	Web Browser
Mkportal	PHP	Yes	Yes	Yes	Web Browser
MMBase	Java	Yes	Yes	Yes	Web Browser
MODx Content Management System	PHP 4/5	Yes	Yes	Yes	Web Browser
NitroTech	PHP	Yes	Yes	Yes	Web Browser
Nucleus CMS	PHP	Yes	Yes	Yes	Web Browser
Nuke-Evolution	PHP	Yes	Yes	Yes	Web Browser
Nuxeo CPS	Zope product	Yes	Yes	Yes	Web Browser
OpenACS	TCL AOLserver	Yes	Yes	Yes	Web Browser
OpenCms	Java	Yes	Yes	Yes	Web Browser
OpenPHPNuke	PHP	Yes	Yes	Yes	Web Browser
PHP-Fusion	PHP	Yes	Yes	Yes	Web Browser
PHP-Nuke	PHP	Yes	Yes	Yes	Web Browser
phpWCMS	PHP	Yes	Yes	Yes	Web Browser
phpWebSite	PHP	Yes	Yes	Yes	Web Browser
phpSlash	PHP	Yes	Yes	Yes	Web Browser
phpCMS	PHP	Yes	Yes	Yes	Web Browser
PhpWiki	PHP	Yes	Yes	Yes	Web Browser
Pivot	PHP	Yes	Yes	Yes	Web Browser
Plone	Zope, Python	Yes	Yes	Yes	Web Browser
PmWiki	PHP	Yes	Yes	Yes	Web Browser
PostNuke	PHP	Yes	Yes	Yes	Web Browser
PuzzleApps	PHP, XML, XSLT	Yes	Yes	Yes	Web Browser
Scoop	Perl on mod perl	Yes	Yes	Yes	Web Browser
Slash	Perl on mod perl	Yes	Yes	Yes	Web Browser

Textpattern	PHP	**Yes**	**Yes**	**Yes**	Web Browser
TikiWiki	PHP	**Yes**	**Yes**	**Yes**	Web Browser
Twiki	Perl	**Yes**	**Yes**	**Yes**	Web Browser
Typo	Ruby on Rails	**Yes**	**Yes**	**Yes**	Web Browser
TYPO3	PHP	**Yes**	**Yes**	**Yes**	Web Browser
UNITED-NUKE	PHP	**Yes**	**Yes**	**Yes**	Web Browser
WebGUI	Perl on mod perl	**Yes**	**Yes**	**Yes**	Web Browser
WordPress	PHP	**Yes**	**Yes**	**Yes**	Web Browser
Xaraya	PHP 4/5 with XHTML/XML /XSLT output	**Yes**	**Yes**	**Yes**	Web Browser
XOOPS	PHP	**Yes**	**Yes**	**Yes**	Web Browser
Zentri	PHP	**Yes**	**Yes**	**Yes**	Web Browser
Modulo	PHP	**Yes**	**Yes**	**Yes**	Web Browser

All of these tools have some limitations:

- Free and open source software;
- The code is not structured and not reused;
- These tools are not oriented patterns;
- The code generated by one specific platform is not used directly on another specific platform (Mobile phone platform to PDA platform).

3. Tools for Model-Based Approach and for Patterns

3.1 Definitions and Advantages of Model-Based UI Development

The model-based approach was introduced to support the specification and design of interactive systems at a semantic, conceptual and abstract level as an alternative to dealing with low-level implementation issues earlier on in the development lifecycle (Seffah and Gaffar, 2006).

The model-based approach uses a central knowledge base to store a description of all aspects of an interface design.

The main function of a model-based approach is to identify useful abstractions and highlight the main aspects that should be considered when designing interactive systems (Sinnig, 2004) and (Marucci et al., 2003). Several models are created and combined to characterize a domain of interest from different perspectives (Sinnig, 2004) and (Forbrig et al., 2003a).

3.1.1 Definition of Model-Based UI Development

That all aspects of a user interface design are represented using declarative models is central to all model-based approaches. The central component is the Interface Model, which includes different declarative models (Sinnig, 2004) and (Schlungbaum, 1996). A series of declarative models, such as user-task, dialog, and presentation, are interrelated to provide a formal representation of an interface design (Sinnig, 2004) and (Puerta, 1997).

In a model-based approach, the UI design is the process of creating and refining the user interface models (Sinnig, 2004) and (Da Silva, 2000). Model-based design focuses on finding the mapping between the various models (Sinnig, 2004) and (Vanderdonckt et al., 2003b). Eventually, User Interfaces are generated automatically or semi-automatically from their model descriptions.

3.1.2 Advantages of Model-Based UI Development

Initially, it may seem that following a model-based approach for the design of interactive systems may complicate and slow down the development process. However, the benefits to be gained are considerable. In essence, model-based UI development has two major advantages (Sinnig, 2004):

Design decisions are made at conceptual levels (i.e. designing the envisioned task model). Designers can specify and analyze interactive systems from a more semantic-oriented level rather than starting immediately to address the implementation level;

- Following a systematic and repeatable development approach to the reconstruction process is easier and affords better comprehension of the system for later maintenance.

3.1.3 Different models

Many facets, as well as related models, exist to describe a User Interface. Until now, there is no agreement on which set of models is best for describing UIs in a declarative manner (Sinnig, 2004) and (Da Silva, 2000). Different model-based approaches use different declarative models (Sinnig, 2004), (Schlungbaum and Elwert, 1996), (Puerta, 1997) and (Vanderdonckt et al., 2003a) (Schlungbaum and Elwert, 1996), (Puerta, 1997) and (Vanderdonckt et al., 2003a). In what follows, the most frequently used models will be defined: user task, user, domain, environment, platform, dialog, and presentation models. These models have different names in different architectures. One should note that some models overlap (as illustrated in Figure 1).

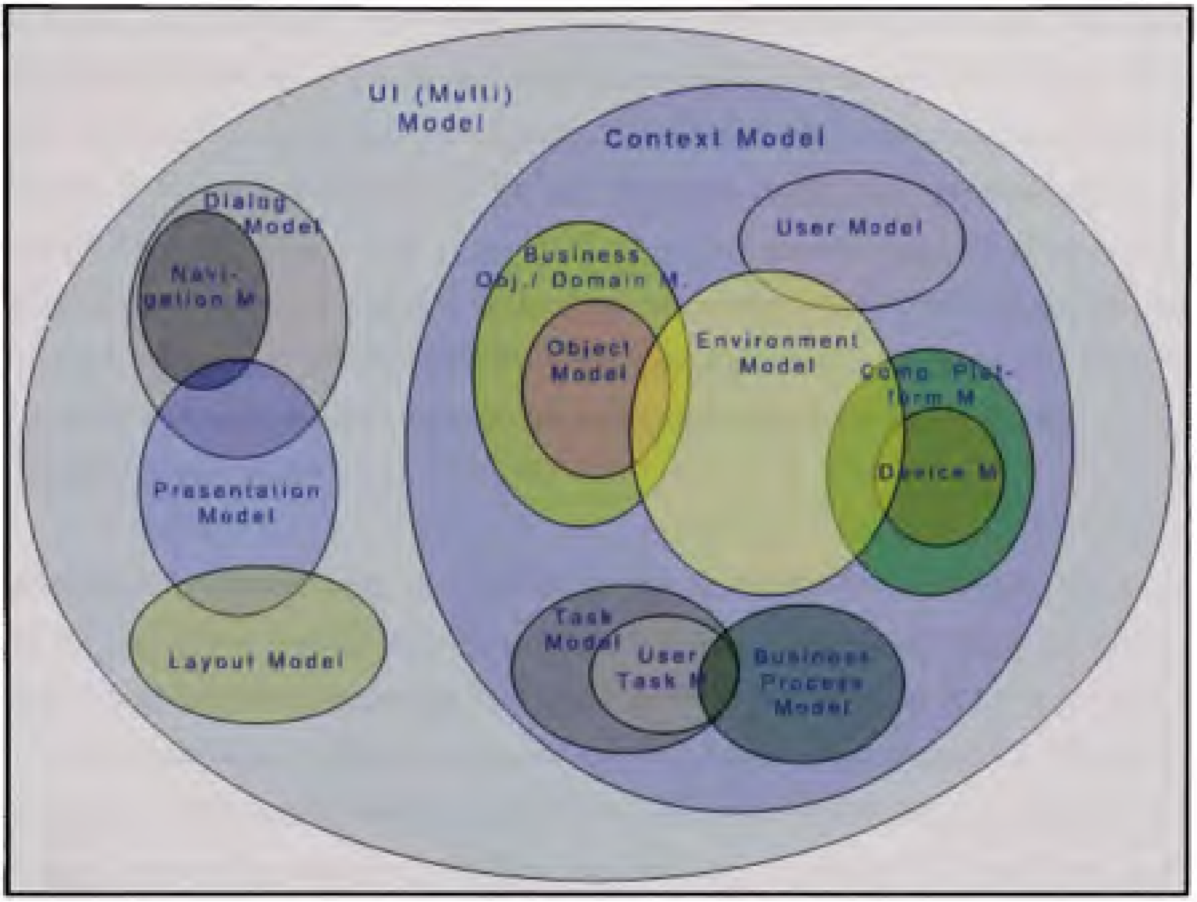

UI (Multi) Model
Context Model
Dialog Model
Navi-gation M.
User Model
Business Obj./ Domain M.
Object Model
Environment Model
Comp. Plat-form M
Device M
Presentation Model
Layout Model
Task Model
User Task M
Business Process Model

Figure 1: Different Model-Based Approaches.
(Extracted from (Sinnig, 2004))

3.2 Different tools for Model-Based Approach

The tool was originally written as a prototype plug-in to the existing **LTSA-WS tool** suite (Beard et al., 1996), providing the groundwork for a Java implementation that collaborated in other extensions to the suite, such as the Message Sequence Chart editor and graphical LTS Draw functions, and which could contribute to future extensions. LTSA uses FSP to specify behavior models. From the FSP description, the tool generates an LTS model. The user can animate the LTS by stepping through the sequences of actions it models, and model-check the LTS for various properties, including deadlock freedom, safety and progress properties. The MSC extension builds by introducing a graphical editor for MSCs and by generating an FSP specification from a scenario description (Vanderdonckt et al., 2003a). An FSP code is generated for the architecture, trace and constraint models described previously. LTSA model checking capabilities (for safety and liveness checks) are then used to detect implied scenarios. The LTSAWS Eclipse plug-in architecture (Figure 2) leverages the previous work and utilizes the model-view-controller pattern. The service implementation model is the BPEL4WS XML source code, and is managed by editing in the form of a standard XML editor. The model is also parsed to provide useful editor functions such as content outline and syntax highlighting. Parsing is also performed on *restore* or *save* actions, whereby the translation function is called to view activities specified in the composition.

CRITIQUE (Paternò, 2005) is a tool that creates KLM/GOMS models from the analysis of the logs of user interactions with graphical interfaces implemented with a research tool. The model is created following two types of rules: the types of KLM operators are identified according to the type of event, and new levels in the hierarchical structure are built when users begin working with a new object or when they change the input to the current object (for example, switching from clicking to typing in a text box). In this approach, the limitation is that the task model only reflects the past use of the system and not other potential uses. These rules for building GOMS models, including mental operators, have then been used in another tool (Paternò, 2005) that analyzes logs of interactions with interfaces. The authors reported that this approach was tested with two users (one was an author) and the models obtained were more accurate than previously published models.

U-Tel (Paternò, 2005) analyses textual descriptions of the activities to support and then to automatically associate tasks with verbs and objects with nouns. **CTTE** (Paternò, 2005) provides the possibility for loading an informal textual description of a scenario or a use case and interactively selecting the information of interest for the modeling work. In this way, the designer can first identify tasks, then create a logical hierarchical structure and finally complete the task model.

The developers of **ISODE** (Paternò, 2005) have considered the success of UML and provide some support to import Use Cases created by **Rational Rose** in their tool for task modeling. This environment also includes **TAMOT,** a tool for modeling tasks specified with the **DIANE**+ notation.

A simulator for task models can be useful to better analyze the dynamic behavior of task models, including those for cooperative systems. This feature is particularly meaningful when the notation used to represent the model allows the specification of many temporal relationships among tasks in addition to sequential tasks (such as disabling tasks, concurrent tasks, suspending tasks). **VTMB** (Paternò, 2005) and **CTTE** are supports that only a few tools provide. Also, in the case of tools for UML, this feature is usually missing.

Java Development Environment Tools (**Eclipse, Rational Rose**) allow one to generate the skeleton of the source code, which is often a badly structured code and sometimes anarchistic. Making it difficult to understand and to reuse.

Eclipse and Rational Rose:

- Plug-Ins (development new tools);
- Beans;
- Pluggable Look & Feel offer a mechanism to move from one model to another. How? By virtual machine to generate a system and to present the system for a specific independent platform (Lunix, Windows, X-Windows);
- Generic development code. Translation from one language to another. There are some problems with this as: 1) the code is not structured like the user wants it, requiring one to decouple the interface and the model, 2) Usability issues since the code generated for a mobile phone platform is not used directly by a PDA platform.

In summary, various solutions are possible for analysis tools based on task modeling. **CTTE** represents a useful contribution to understanding the possibilities in terms of the analyses that can be performed. One sees that CTTE is also able to compare two models with respect to a set of metrics. **Euterpe** also supports the calculation of some metrics to analyze a specification, and to help find inconsistencies or problems. The ability to predict task performance is usually supported by tools for GOMS such as **QDGOMS** (Beard et al., 1996). Overall, one important feature is the possibility of interactively simulating the task model's dynamic behavior (Paternò, 2004).

This following table summarizes the weaknesses of these tools on the level of the interactive system's development based patterns.

Criteria Tools	Analyzing Models	Designing Models	Generation code	Open source	Pattern-Oriented	Model-Based
LTSA-WS	Yes	Yes	Yes	Yes	No	No
CRITIQUE	Yes	Yes	Yes	Yes	No	No
U-TEL	Yes	Yes	Yes	Yes	No	No
CTTE	Yes	No	No	Yes	No	No
ISODE	Yes	Yes	Yes	Yes	No	No
TAMOT	Yes	Yes	Yes	Yes	No	No
DIANE	Yes	Yes	Yes	Yes	No	No
VTMB	No	Yes	Yes	Yes	No	No
ECLIPSE	Yes	Yes	Yes	Yes	No	Partially
RATIONAL ROSE	Yes	Yes	Yes	No	No	Yes
TIDE 2	No	Yes	Yes	Yes	Partially	Yes

3.3 Pattern-Oriented Tools

A tool support for pattern-oriented design should enhance the pattern user's understandability, decrease the complexity of a pattern, and eliminate terminological ambiguity. At the same time, the pattern language should be put into practice in a real context of use, which is a critical issue for making pattern languages a cost-effective vehicle for the gathering and disseminating of the best design practices among system and usability engineering teams.

4. Formalisms and notations for patterns, architectures and models specifications

4.1 Different formalisms and languages

The majority of the languages are:

- Inappropriate;
- Difficult to use in an industrial context;
- Focused on an aspect of coordination of activities and interoperability;
- Temporal synchronization, resource sharing, collaboration between individuals;
- Focused on a component of the software (resource, activity, tool, and person).

In the majority of the architectures studied:

- POD, PSA, MDA architectures are incomplete in the sense that each one of these architectures does not take into account the important concepts of the other for the development of system;
- There are no use patterns in interactive systems;
- Relationships between the patterns do not exist.

The languages and formalisms of existing patterns:

1. **In the case of implementation :**
    - Allow one to model software architectures under specific aspects;
    - Are not easily usable in a real context;
    - Make it possible to describe only very specific software;
    - Adaptation is very difficult.
2. **In the case of Design :**
    - No formalism or language is offered;
    - These tools are not easy to use by nonprogrammers;

- These tools do not support a dynamic approach (Dawayne, 1993);
- These tools do not provide an interactive and graphic interface (Carr et al., 1995).

A number of pattern languages have been suggested. For example, Van Duyne's (2003) "The Design of Sites", Welie's (1999) Interaction Design Patterns, and Tidwell's (1997) UI Patterns and Techniques play an important role and Roberts's et al. (2001) Designing for the User with OVID: Bridging User Interface Design and Software Engineering, graphical and visual notation and method Design Approach. In addition, specific languages such as Laakso's (2003) User Interface Design Patterns and the UPADE Language (Engelberg and Seffah, 2002) have been proposed as well.

4.2 Examples of different languages and notations

4.2.1 User Interface Markup Language (UIML)

UIML is a meta-language that allows the developer to describe the user interface (UI) in generic terms and to use style descriptions to map the UI to various target platforms. UIML was developed to address the need for a uniform UI description language for building multi-platform systems.

A UIML document contains three different parts: a UI description, a peers section that defines mappings from the UIML document to external entities (rendering to the target platform and system logic), and finally a template section that allows the reuse of written elements. The UI description specifies a set of interface elements with which the end user interacts. For each element, a presentation style is defined (e.g. position, font, color) along with its content, possible user input events, and resulting actions.

Eventually, a UI description can be rendered to the corresponding target platform, resulting in a functional UIML that provides a uniform language to describe user interfaces for different target platforms. However, the creation of user interfaces for different target platforms from a single specification is not possible. There is still a need to design separate user interfaces for each device (Sinnig, 2004).

4.2.2 eXtensible User Interface Language (XUL)

XUL (Sinnig, 2004), (XUL, 2004a) and (XUL, 2004b) is an official Mozilla initiative, and provides an XML-based language for describing window layout. The goal of XUL is to build cross platform systems, which are easily portable to all of the operating systems on which Mozilla runs. XUL provides a clear separation between the user interface definition (the various widgets that determine the UI) and its visual appearance (the layout and the "look and feel"). A UI is described as a set of structured interface elements along with a set of predefined attributes. Event handlers and scripts can be defined in order to allow interaction with the user. In order to extend XUL, the XBL (eXtensible Bindings Language) (Sinnig, 2004) and (XBL, 2004) can be used to define new elements and XUL widgets. In addition, it is possible to integrate external libraries (i.e. written in C/C++ or JavaScript) using the XPCOM / XPConnect interfaces.

However, XUL has its focus on window-based user interfaces. This focus has a limitation. At the moment, XUL specifications cannot be rendered to multiple user interfaces, including small mobile devices (Sinnig, 2004) and (Souchon and Vanderdonckt, 2003).

4.2.3 eXtensible Interface Markup Language (XIML)

XIML is the follower of MIMIC (Sinnig, 2004) and (Puerta and Maulsby, 1997) and provides a way to describe the UI without worrying about its implementation. The goal of XIML is to describe the various abstract (domain, task, and user) and concrete (presentation and dialog) aspects of the UI throughout the development lifecycle. In addition, XIML supports the definition of mapping from abstract to concrete elements (Sinnig, 2004) and (Puerta and Eisenstein, 1999).

Figure 2 illustrates the basic structure of XIML. Practically, it is a hierarchical organized set of interface elements that are distributed to one or more interface components. One should note that XIML uses the term 'component' instead of 'model'. XIML predefines five basic interface components:

- **Task component**: Captures the business process and/or user tasks that the interface supports;
- **Domain component**: Comprises a set of all objects and classes used;
- **User component**: Captures the characteristics of the users of the system;
- **Dialog component**: Specifies the UI interaction;
- **Presentation component**: Defines a hierarchy of concrete interaction objects.

However, the language does not limit the number and types of components and elements. XIML can be extended in order to accommodate customized or new interface components. In addition to the interface components, a XIML (Sinnig, 2004) and (XIML, 2003) description consists also of attributes and relations. On one hand, an attribute is a feature or property that has a value and belongs to an element. On the other hand, a relation is used in order to link one or more elements together within the same component or across several components.

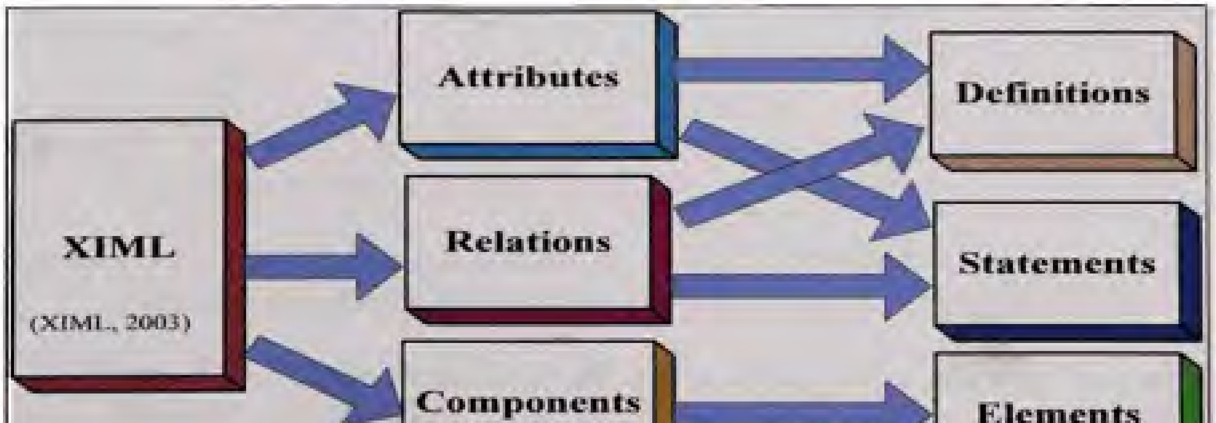


Figure 2: The Basic Structure of XIML.

XIML (Sinnig, 2004) has been introduced mainly to standardize the representation of different models in order to act as a universal exchange format and to foster the interoperability of systems. However, there are currently only a few tools such as Vaquita (Bouillon and Vanderdonckt, 2002), XIML-Task-Simulator (Forbrig et al., 2003b), and Dialog-Graph-Editor (Forbrig et al., 2003b) which use XIML. Therefore, there are no tools presently available which are capable of rendering a XIML description to a user interface. In addition, the extensibility of XIML also has its limitations. By defining new XIML Components, the interoperability cannot be ensured. If portability is needed, XIML Components may need to be limited to boundaries that are predefined.

4.2.4 Existing Model-Based Framework

Model-based UI development (Sinnig, 2004) has been investigated for more than a decade. Many groups and individuals have devoted themselves to the development of model-based frameworks. This section gives an overview of the most current and influential approaches. One should note that instead of automation, **JANUS** (Balzert, 1996), **AME** (Märtin, 1996), "modern" model-based systems **MOBI-D** (1999), **TERESA** (2004) provide tools that allow developers to interactively define mappings between the various models.

5. References

Balzert, H. 1996. From OOA to GUIs: The JANUS System. *Journal of Object-Oriented Programming*. (February, 1996). pp. 43-47.

Beard, D., D. Smith, K. Denelsbeck. 1996. Quick and Dirty GOMS: A Case Study of Computed Tomography. *Human-Computer Interaction*. V.11, N.2, pp.157-180.

Bouillon, L. and J. Vanderdonckt. 2002. Retargeting of Web Pages to Other Computing Platforms with VAQUITA. In Proceedings of WCRE'02. October 2002. Richmond (Virginia).

Carr, David C. and Ashok Dandekar and Perry E. Dawayne. 1995. Experiments in Process Interface Description, Visualizations and Analyses, Software Process, Technology. Fourth European Workshop – EWSPT'95. Springer-Verlag.

Da Silva, P. 2000. User Interface Declarative Models and Development Environments: A Survey. In Proceedings of DSV-IS'2000. pp. 207–226: Springer.

Duyne, D. K., van J. A. Landay and J. I. Hong. 2003. *The Design of Sites: Patterns, Principles and Processes for Crafting a Customer-Centered Web Experience.* Addison Wesley.

Engelberg, D., and A. Seffah. 2002. A. Design Patterns for the Navigation of Large Information Architectures. 11th Annual Usability Professional Association Conference. Orlando: (FL). USA

Forbrig, Peter, A. Dittmar and A. Mueller. 2003a. Adaptive Task Modelling: From Formal Models to XML Representations. Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces. pp. 171-192. London. Welie.

Forbrig, Peter, A. Dittmar, D. Reichart and D. Sinnig. 2003b. User-Centred Design and Abstract Prototypes. In Proceedings of BIR 2003. pp. 132 – 145. SHAKER. Berlin (Germany).

Grundy, John and Wenjing Zou. 2004. AUIT: Adaptable User Interface Technology, with extended Java Server pages', Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces. London. Wiley.

Roberts, Dave, Dick Berry, Scott Isensee, and John Mullaly. 2001. Designing for the User with OVID: Bridging User Interface Design and Software Engineering. Published by Prentice Hall 2001, 400 pages. ISBN 013092377X.

Laakso, Sari A. 2003. Collection of User Interface Design Patterns. University of Helsinki, Dept. of Computer Science.

Märtin, C. 1996. Software Life Cycle Automation for Interactive Applications: The AME Design Environment. In *Proceedings of CADUI'96.* June 1996. pp. 57-76. Namur (Belgium): Namur University Press.

Marucci, L., F. Paternó and C. Santoro. 2003. Supporting Interactions with Multiple Platforms Through User and Task Models. Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces. pp. 217-238. London. Wiley.

MOBI-D. 1999. *The MOBI-D Interface Development Environment*. Online. <http://smi-web.stanford.edu/projects/mecano/mobi-d.htm>. Accessed on December 8th 2006.

Paternò, Fabio. 2004. Model-based Tools for Pervasive Usability. ENC 2004: 6.

Paternò, Fabio. 2005. Model-Based Tools for Pervasive Usability. Interacting with Computers 17(3). Pages 291-315.

PhPNuk. 2006. Online. <http://phpnuke.org/>. Accessed on September 20th 2006.

Puerta, A. 1997. A Model-Based Interface Development Environment. IEEE Software 0740 – 7459 / 97, Vol. 14, pp. 41-47. Online. <http://www.arpuerta.com/pdf/ieee97.pdf>. Accessed on October 23rd 2005.

Puerta, A. and D. Maulsby. 1997. Management of Interface Design Knowledge with MODI-D. In Proceedings of IUI'97. January 1997. pp. 249-252. Orlando (FL).

Puerta, A. and J. Eisenstein. 1999. Towards a General Computational Framework for Model-Based Interface Development Systems. In Proceedings of IUI'99. 5-8 January 1999. pp. 171-178. Redondo Beach (CA): ACM Press. New York (USA).

Schlungbaum, E. 1996. Model-Based User Interface Software Tools – Current State of Declarative Models. Technical Report 96-30. Graphics, Visualization and Usability Center Georgia Institute of Technology.

Schlungbaum, E. and T. Elwert. 1996. Automatic User Interface Generation from Declarative Models. In Proceedings of CADUI 96. pp. 3-18. Namur (Belgium): Namur University Press.

Seffah, Ahmed and Gaffar, Ashraf. 2006. Model-based user interface engineering with design patterns. Journal of Systems and Software, doi:10.1016/j.jss.2006.10.037, 15 pages.

Sinnig, Daniel. 2004. The complicity of patterns and Model-Based UI Development. Master of Computer Science, Montreal, Concordia University. 161 p.

Souchon, N. and J. Vanderdonckt. 2003. A Review of XML-compliant User Interface Description Languages. In Proceedings of DSV-IS 2003. pp. 377-391. Funchal (Portugal).

TERESA. 2004. *Transformation Environment for Interactive Systems Representations*. Online. <http://giove.cnuce.cnr.it/teresa.html>. Accessed on December 15th 2006.

Tidwell, J. Common Ground. 1997. A Pattern Language for Human-Computer Interface Design. Online. <http ://www.mit.edu/~jtidwell/common_ground.html>. Accessed on October 5th 2005.

Vanderdonckt, J., E. Furtado, J. Furtado and Q. Limbourg. 2003a. Multi-Model and Multi-Level Development of User Interfaces. *Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces*. pp. 193-216. London. Wiley.

Vanderdonckt, J., Q. Limbourg and M. Florins. 2003b. Deriving the Navigational Structure of a User Interface. In Proceedings of INTERACT 2003. September 2003. IOS. pp. 455-462. Zurich (Switzerland).

Vanderdonckt, J., E. Furtado, J. Furtado and Q. Limbourg. 2003a. Multi-Model and Multi-Level Development of User Interfaces. *Multiple User Interfaces, Cross-Platform Applications and Context-Aware Interfaces*. pp. 193-216. London. Wiley.

Welie, M.V. The Amsterdam. 1999. Collection of Patterns in User Interface Design. Online. <http://www.cs.vu.nl/~martijn/patterns/index.html>. Accessed on October 2nd 2005.

Wikipedia. The free encyclopedia. Content Management System. Online. <http://en.wikipedia.org/wiki/Content_management_system>. Accessed on Mars 15th 2006.

XBL. 2004. The eXtensible Binding Language 1.0. Online. <http://developer.mozilla.org/en/docs/XBL:XBL_1.0_Reference>. Accessed on November 14th 2006

XIML. 2003. eXtensible Interface Markup Language. Online. <http://www.ximl.org/documents/XimlWhitePaper.pdf>. Accessed on November 15th 2006.

XUL. 2004a. The XML User Interface Language. Online. <http://www.xulplanet.com/>. Accessed on November 28th 2006.

XUL. 2004b. XUL Tutorial. Online. <http://www.xulplanet.com/tutorials/xultu/>. Accessed on December 2nd 2006.

Zope. 2008. Online. <http://www.zope.org>. Accessed on January 10th 2008.

APPENDIX III

EXAMPLE OF XML SOURCE CODE FOR POMAML STRUCTURAL NOTATION

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by cordier (none) -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="POMAML">
    <xs:complexType>
      <xs:choice>
        <xs:element name="Patterns">
          <xs:complexType>
            <xs:sequence maxOccurs="6">
              <xs:annotation>
                <xs:documentation>Patterns Composition Rules</xs:documentation>
              </xs:annotation>
                <xs:element name="NavigationPatterns" type="TaskPatternType"/>
                <xs:element name="InteractionPatterns" type="TaskPatternType"/>
                <xs:element name="PresentationPattern" type="TaskPatternType"/>
                <xs:element name="VisualizationPatterns" type="TaskPatternType"/>
                <xs:element name="InteroperabilityPatterns" type="TaskPatternType"/>
                <xs:element name="InformationPatterns" type="TaskPatternType"/>
                <xs:element name="PIMModels">
                  <xs:complexType>
                      <xs:sequence maxOccurs="5">
                        <xs:annotation>
                          <xs:documentation>Models Transformation
Rules</xs:documentation>
                        </xs:annotation>
                            <xs:element name="PIMDomainModel"
type="PIMDomainModelType"/>
                            <xs:element name="PIMTaskModel" type="PIMTaskModelType"/>
                            <xs:element name="PIMDialogModel" type="PIMDialogModelType"/>
                            <xs:element
name="PIMPresentationModel"type="PIMPresentationModelType"/>
                            <xs:element name="PIMLayoutModel"
type="PIMLayoutModelType"/>
                      <xs:sequence>
                      <xs:annotation>
                            <xs:documentation>Patterns Mapping Rules</xs:documentation>
                      </xs:annotation>
```

```
                    <xs:element name="PSMModels">

                      <xs:complexType>
                            <xs:sequence>
                          <xs:annotation>
                                <xs:documentation>Models Transformation
Rules</xs:documentation>
                          </xs:annotation>
                                <xs:element name="PSMDomainModel"
type="PSMDomainModelType"/>
                              <xs:element name="PSMTaskModel" type="PSMTaskModelType"/>
                              <xs:element name="PSMDialogModel" type="PSMDialogModelType"/>
                        <xs:element name="PSMPresentationModel"
type="PSMPresentationModelType"/>
                              <xs:element name="PSMLayoutModel" type="PSMLayoutModelType"/>
                        <xs:sequence>
                          <xs:annotation>
                          <xs:documentation>Source Code Generation FRules</xs:documentation>
                          </xs:annotation>
                         <xs:element name="ApplicationGenaration"
type="ApplicationGenarationType"/>
                        </xs:sequence>
                    </xs:sequence>

        </xs:complexType>

        </xs:element>
                                                                      </xs:sequence>
                                                                 </xs:sequence>
                                                            </xs:complexType>
                                                       </xs:element>
                                                  </xs:sequence>
                                             </xs:complexType>
                                        </xs:element>
                                   </xs:choice>
                              </xs:complexType>
                         </xs:element>
        <xs:complexType name="TaskType">
             <xs:sequence>
                  <xs:element name="ID" type="xs:string"/>
                  <xs:element name="Name" type="xs:string"/>
                  <xs:element name="Order" type="xs:string"/>
                  <xs:element name="Relation" type="xs:string" minOccurs="0"
maxOccurs="unbounded"/>
                  <xs:element name="SubTasks" minOccurs="0">
```

```
                <xs:complexType>
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="Task">
                            <xs:complexType>
                                <xs:complexContent>
                                    <xs:extension
base="TaskType"/>
                                </xs:complexContent>
                            </xs:complexType>
                        </xs:element>
                        <xs:element name="SubTasks"
type="TaskTemplateType"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TaskTemplateType">
        <xs:sequence>
            <xs:element name="ID" type="xs:string"/>
            <xs:element name="VariableDef" minOccurs="0" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name="Name" type="xs:string"/>
                        <xs:element name="Description" type="xs:string"/>
                    </xs:sequence>
                </xs:complexType>
            </xs:element>
            <xs:element name="Name" minOccurs="0">
                <xs:complexType>
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="Text" type="xs:string"/>
                        <xs:element name="Variable" type="xs:string"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
            <xs:element name="Order" type="xs:string"/>
            <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="SubTasks" minOccurs="0">
                <xs:complexType>
                    <xs:choice minOccurs="0" maxOccurs="unbounded">
                        <xs:element name="Task" type="TaskType"/>
                        <xs:element name="Subtasks"
type="TaskTemplateType"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="TaskPatternType">
        <xs:sequence minOccurs="0" maxOccurs="unbounded">
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Problem" type="xs:string"/>
            <xs:element name="Context" type="xs:string"/>
            <xs:element name="Solution" type="xs:string"/>
            <xs:element name="Rational" type="xs:string"/>
            <xs:element name="Body">
                <xs:complexType>
                    <xs:choice>
                        <xs:element name="Task" type="TaskType"/>
                        <xs:element name="TaskTemplate"
type="TaskTemplateType"/>
                    </xs:choice>
                </xs:complexType>
            </xs:element>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="ApplicationGenarationType">
        <xs:choice>
            <xs:element name="Language">
                <xs:complexType>
                    <xs:sequence/>
                </xs:complexType>
            </xs:element>
        </xs:choice>
    </xs:complexType>
    <xs:complexType name="PSMDomainModelType">
        <xs:sequence>
            <xs:element name="Name"/>
            <xs:element name="Descrikption"/>
            <xs:element name="Example"/>
            <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PSMTaskModelType">
        <xs:sequence>
            <xs:element name="Name"/>
            <xs:element name="Description"/>
            <xs:element name="Example"/>
            <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PSMDialogModelType">
        <xs:sequence>
            <xs:element name="Name"/>
            <xs:element name="Description"/>
            <xs:element name="Example"/>
            <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PSMPresentationModelType">
        <xs:sequence>
            <xs:element name="Name"/>
            <xs:element name="Description"/>
            <xs:element name="Example"/>
            <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PSMLayoutModelType">
        <xs:sequence>
            <xs:element name="Name"/>
            <xs:element name="Description"/>
            <xs:element name="Example"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PIMDomainModelType">
        <xs:sequence>
            <xs:element name="Name" type="xs:string"/>
            <xs:element name="Description" type="xs:string"/>
            <xs:element name="Example" type="xs:string"/>
            <xs:element name="Relation" minOccurs="0" maxOccurs="unbounded"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="PIMTaskModelType">
        <xs:complexContent>
            <xs:extension base="TaskModelType">
                <xs:sequence>
                    <xs:element name="Name"/>
                    <xs:element name="Description"/>
                    <xs:element name="Example"/>
                    <xs:element name="Relation" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="PIMDialogModelType">
        <xs:complexContent>
            <xs:extension base="DialogModelType">
                <xs:sequence>
                    <xs:element name="Name"/>
                    <xs:element name="Description"/>
                    <xs:element name="Example"/>
                    <xs:element name="Relation" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="PIMPresentationModelType">
        <xs:complexContent>
            <xs:extension base="PresentationModelType">
                <xs:sequence>
                    <xs:element name="Name"/>
                    <xs:element name="Description"/>
                    <xs:element name="Example"/>
                    <xs:element name="Relation" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
    <xs:complexType name="PIMLayoutModelType">
        <xs:complexContent>
            <xs:extension base="LayoutModelType">
                <xs:sequence>
                    <xs:element name="Name"/>
                    <xs:element name="Description"/>
                    <xs:element name="Example"/>
                    <xs:element name="Relation" minOccurs="0"
maxOccurs="unbounded"/>
                </xs:sequence>
            </xs:extension>
        </xs:complexContent>
    </xs:complexType>
</xs:schema>
```

LIST OF APPENDICES

Page

APPENDIX I GLOSSARY OF TERMS 151

APPENDIX II TECHNICAL REPORT OF INTERACTIVE SYSTEM DEVELOPMENT TOOLS: TRENDS AND CHALLENGES IN INTERACTIVE SYSTEM DEVELOPMENT TOOLS: REQUIREMENTS FOR PATTERN-ORIENTED AND MODEL-BASED ARCHITECTURE 145

APPENDIX III EXAMPLE OF XML SOURCE CODE FOR POMAML STRUCTURAL NOTATION……………………………………………172

BIBLIOGRAPHY

Alexander, Christopher. 1979. *The Timeless Way of Building*. Oxford University Press. New York: (NY). USA.

Alexander, Christopher, S. Ishikawa, M. Silverstein, M. Jacobson, I. Fiskdahl-King and S. Angel. 1977. A Pattern Language. Oxford University Press. New York. USA.

Bass, Len, Paul Clements and Rick Kazman. 2003. *Software Architecture in Practice*. Second Edition. Addison-Wesley Boston: (MA).USA.

Beck, K., J. O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. Vlissides. 1996. Industrial experience with design patterns. In Proceedings of the 18th International Conference on Software Engineering. Pages 103–114. IEEE Computer Society Press.

Booch, Grady, James Rumbaugh and Ivar Jacobson. 1999. *The Unified Modeling Language User Guide*. Addison-Wesley.

Buschmann, F. 1996. What is a pattern? Object Expert. Vol. 1(3). PP17-18.

Buschmann, F., R. Meunier, Rohnert H., P. Sommerlad and M. Stal. 1996. *Pattern-Oriented Software Architecture: A System of Patterns*. John Wiley & Sons Ltd.

Carr, David C. and Ashok Dandekar and Perry E. Dawayne. 1995. Experiments in Process Interface Description, Visualizations and Analyses, Software Process, Technology. Fourth European Workshop – EWSPT'95. Springer-Verlag.

CHI. 1999. Human Factors in Computing Systems. Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit. Online. <http://portal.acm.org/toc.cfm?id=302979&coll=ACM&dl=ACM&type=proceeding&idx=SERIES260&part=Proceedings&WantType=Proceedings&title=Conference%20on%20Human%20Factors%20in%20Computing%20Systems&CFID=78433578&CFTOKEN=67946859>. Pittsburgh: (Pennsylvania). USA. Accessed on January 5th 2007.

Clarke, S. 2000. Composing Design Models: An Extension to the UML. UML 2000 - International conference on the unified modeling language N°3. York. Royaume-Uni. (02/10/2000) 2000. Vol. 1939. Pages 338-352. [Note(s): XIV, 572 p.,] (9 ref.). **ISBN** 3-540-41133-X.

Clarke, S. and R. Walker. 2001. Composition Patterns. An Approach to Designing Reusable Aspects. To appear in Proc. ICSE.

Coad, Peter. 1992. Object Oriented Patterns. Communications of the ACM. Vol. 35. No: 9.

Coplien, J. O. 1998. *The Patterns Handbook: Techniques, Strategies, and Applications*, chapter Software Design Patterns: Common Questions and Answers. Pages 311–320. Cambridge University Press. New York: (NY). USA.

Coram, T. and J. Lee. 1998. A Pattern Language for User Interface Design. Online. <http://www.maplefish.com/todd/papers/experiences>. Accessed on October 1st 2005.

Coutaz, J. 1987. PAC, an implementation Model for dialog Design. Interact'87. pp. 431-436. Stuttgart. Germany.

Coutaz, J. 1990. Architecture Models for interactive software: Failures and trends. In Engineering for Human-Computer Interaction. Cockton G. Ed. Elsevier Science Publication. pp 137-153.

Da Silva, P. P. 2000. User Interface Declarative Models and Development Environments: A Survey. Proceedings of Seventh International Workshop on Design. Specification and Verification of Interactive Systems (DSVIS 2000). Limerick. Ireland.

Darses, F. 1990. Constraints in design: towards a methodology of psychological analysis based on AI formalisms. In Proceedings of conference on Human-Computer Interaction. INTERACT'90. pp 135-138. Diaper et al. D. (Eds). Elsevier Science Publishers B. V. North Holland.

Dawayne, Perry E., 1993. Human in the Process: Architectural Implications. Proceeding of the 8th International Software Process Workshop. Schloss Dagstuhl. Germany.

Donyaee, Mohammad K. 2008. Investigating the Correlation of Usability Measures and User Tests: A Roadmap for a Predictive Model. PhD of Computer Science and Software Engineering, Montreal, Concordia University, 201 p.

D'Souza, Desmond. 2001. Model-Driven Architecture and Integration Opportunities and Challenges. OMG Group. Online. <ftp://ftp.omg.org/pub/docs/ab/01-03-02.pdf>. Accessed on September 7th 2005.

Duyne, D. K. van, J. A. Landay, and J. I. Hong. 2003. *The Design of Sites: Patterns, Principles and Processes for Crafting a Customer-Centered Web Experience*. Addison Wesley.

Engelberg, D. and A. Seffah. 2002. A Design Patterns for the Navigation of Large Information Architectures. 11th Annual Usability Professional Association Conference. Orlando (Florida).USA.

Erickson, T. 2000. Lingua Franca for Design: Sacred Places and Pattern Language. In *Proceedings of Designing Interactive Systems*. ACM Press. New York: (NY). USA.

Fowler, Martin. 1997. *Analysis Patterns, reusable objects models.* Addison-Wesley.

Gamma, E., R. Helm, R. Johnson and J. Vlissides. 1995. *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison Wesley.

Garrido, A., G. Rossi and D. Schwabe. 1997. Pattern Systems for Hypermedia. Pattern Language of Programming Conference.

Garrigós, Irene, Jaime Gómez and Cristina Cachero. 2003a. Modelling Adaptive Web Applications. ICWI 2003. pp 813-816.

Garrigós, Irene, Jaime Gómez and Cristina. Cachero. 2003b. Modelling Dynamic Personalization in Web Applications. ICWI 2003. pp 813-816.

Goldberg, A. 1984. Smaltalk-80: The Interactive Programming Environment. Addison-Wesley Publication.

Gram, C. and G. Cockton (eds). 1996. *Design Principles for Interactive Software.* Chapman & Hall.

Granlund, Åsa, Daniel Lafrenière and David A. Carr. 2001. A Pattern-Supported Approach to the User Interface Design Process. Proceedings of HCI International 2001 9th International Conference on Human-Computer Interaction. August 5-10, 2001. New Orleans. USA.

Green, M. 1985. User Interface Management System. ACM. Volume 19. No.3. The University of Alberta. Alberta. Canada.

Henderson-Sellers, Brian, Ian M. Graham, P. Swatman, Russel L. Winder and Trygve Reenskaug. 1996. Using Object-Oriented Techniques to Model the Lifecycle for OO Software Development. OOIS 1996: 211.

Horton, W. K. 1994. *Designing and Writing Online Documentation.* 2nd edition. Wiley. New York: (NY). USA.

IBM. 2007. Design Principles Checklist. Online. <https://www-306.ibm.com/software/ucd/designconcepts/designbasics.html>. Accessed on January 15th 2008.

IEEE. 2000. Recommended Practice for Architectural Description of Software-Intensive Systems IEEE Standard 1471-2000.

IFEN. Institut Français de l'ENvironnement. Ministère de l'Écologie, de l'Énergie, du Développement durable et de l'Aménagement du territoire. Online. <http://www.ifen.fr/acces-thematique/>. France. Accessed on January 15th 2007.

INTERACT. 1999. Technology and Persons with Disabilities. Proceedings of the Center on Disabilities. Online. <http://www.csun.edu/cod/conf/1999/proceedings/csun99.htm>. Edinburgh: (Glasgow). UK. Accessed on February 10th 2006.

ISO/IEC 9126-1:2001. 2001. Software Engineering – Product Quality – Part 1: Quality Model. ISO.

Javahery, H., A. Deichman, A. Seffah, and M. Taleb. 2007. *A User-Centered Framework for Deriving a conceptual design from user experiences. Leveraging personas and patterns to create usable design.* In A. Seffah, J. Gulliksen, and M. Desmarais, eds, Human-Centered Software Engineering, Volume II, Software Engineering Models, Patterns and Architectures for HCI, Chapter 4, May 28th 2007, John Wiley & Sons, New York, USA.

Javahery, H. and A. Seffah. 2002. A Model for Usability Pattern-Oriented Design. In *Proceedings of TAMODIA 2002*. Bucharest. Romania. pp. 104-110.

Javahery, H. 2003. Pattern-Oriented Design for Interactive Systems. Master of Computer Science and Software Engineering, Montreal, Concordia University, 104 p.

Jerome, Bill, Rick Kazman. 2007. Two Studies: Human Computer Interaction and Software Engineering, a chapter in Human-Centered in Software Engineering – Integrating Usability in the Software Development Lifecycle. Edited by Seffah, A., Gulliksen, J. Desmarais, M, Springer, ISBN: 978-1-4020-4027-6.

Johnson, R.E. 1997. Frameworks = Components + Patterns. Communication of the ACM. Vol. 40. No10. pp 39-42.

Keller, Rudolf K. and Reinhard Schauer. 1998. Design Components: Towards Software Composition at the Design Level. ICSE 1998. pp 302-311.

Koch, Nora and Andreas Kraus. 2003. Towards a Common Metamodel for the Development of Web Applications. ICWE 2003. pp 497-50.

Koch, Nora and GmbH Fast. 2006. Transformation Techniques in the Model-Driven Development Process of UWE. ICWE'06 Workshops. Palo Alto: (CA). USA.

Kristensen, Bent Bruun and Kasper Østerbye. 1996. A Conceptual Perspective on the Comparison of Object-Oriented Programming Languages. SIGPLAN Notices 31(2). pp 42-54.

Laakso, Sari A. 2003. Collection of User Interface Design Patterns. University of Helsinki, Dept. of Computer Science.

Larsen, G. 1999. Designing Component-Based framework Using Patterns in the UML. Communication of the ACM. Vol. 42. No 10. pp 38-45.

Lauder, Anthony and Stuart Kent. 1998. Precise Visual Specification of Design Patterns. ECOOP 1998. pp 114-134.

Lea, D. 1997. Patterns Discussion FAQ. Online. <http://g.oswego.edu/dl/pd-FAQ/pd-FAQ.html>. Accessed on October 5th 2005.

Loureiro, K. and D. AD. Plummer. 1999. Patterns: Beyond Objects and Components. Research Note # COM-08-0111. Gartner Group.

Lynch, P.J. and S. Horton. 1999. Web Style Guide: Basic Design Principles for Creating Web Sites. New Haven and London. Yale University Press.

Macintosh. 1992. Human Interface Guidelines. Apple Computer Company. Publisher Addison Wesley Professional. Cupertino: (CA). USA. Online. <http://interface.free.fr/Archives/Apple_HIGuidelines.pdf>. Accessed on February 3rd 2007.

Meservy, Thomas O. and Kurt D. Fenstermacher. 2005. Transforming Software Development: An MDA Road Map. IEEE Computer. Vol. 38. No. 8. pp. 52-58.

Meyer, Bertrand. 1990. *Conception et programmation par objets pour du logiciel de qualité*. Informatique intelligence artificielle ISSN 0297-5416. Source [Note(s) : [622 p.]] ISBN 2-7296-0272-0. Inter-Éditions. Paris. France.

Microsoft. 1995. The Windows Interface Guidelines for Software Design. Microsoft Press. Redmond: (WA). USA. Online. <http://www.ics.uci.edu/~kobsa/courses/ICS104/course-notes/Microsoft_WindowsGuidelines.pdf>. Accessed on February 4th 2007.

Miller, J., J. Mukerji. 2003. MDA Guide Version 1.0.1. OMG doc.omg/2003-06-01. Online. <http://www.omg.org/docs/omg/03-06-01.pdf>. Accessed on September 7th 2005.

Msheik, Hamdan, Alain Abran and Éric Lefebvre. 2004. Compositional Structured Component Model: Handling Selective Functional Composition. IEEE 30th EUROMICRO Conference. pp 74-81.

Mukerji, Jishnu. 2001. Model Driven Architecture (MDA) – Technical Perspective. Document number ormsc/2001-07-01. Architecture Board. ORMSC. 2001. OMG Group. Online. <http://www.omg.org/docs/omg/01-07-01.pdf>. Accessed on September 7th 2005.

Muller, Pierre-Alain and Gartner Nathalie. 2000. *Modélisation Objet avec UML*. Éditions Eyrolles. Paris. France.

Myers. B. A. 1986. Visual programming, programming by example, and program visualization: A taxomany. In Proceedings of the ACM CHI'86 Conference on Human Factors in Computing Systems. ACM New York. pp 271-278.

Myers, B. A. and B W. Buxton. 1986. "Creating highly-interactive and graphical UIs by demonstration", Proceedings of the 13th annual International Conference on Computer Graphics and Interactive Techniques, pp. 249-258.

Myers, B. A. 1989a. User Interface Tools: Introduction and Survey. IEEE Software.

Myers, B. A. 1989b. The sate of the art in visual programming. In Kilgour A. & Earnsnaw R. (Ed) Graphics Tools for Software Engineers. Cambridge University Press.

Myers B. A. 1989c. Encapsulating interactive behaviours. Proceedings of the conference on human factors in computing system (SIGCHI 89). pp 319-324.

Myers, B. A. 1990. Taxonomies of visual Programming and Program Visualization. Journal of Visual Languages and Computing. 1 (1).

Myers, B. A., D.A. Guise, R. B. Dannenberg, B. V. Zanden, D. S. Kosbie, E. P. Mickish, A. P. Marchal, and N. Garnet. 1990. Comprehensive Support for Graphical. Highly Interactive User Interfaces. IEEE Computer. 23(11).

Nielsen, J. 1999. Designing Web Usability. The Practice of Simplicity. New Riders.

OMG Group. 2008. Model-Driven Architecture Home Page. Online. <http://www.omg.org/mda/index.htm>. Accessed on September 1st 2005.

ORMSC White Paper. Ormsc/05-04-01. 2005. A Proposal for an MDA Foundation Model. V00-02. OMG Group. Online. <http://www.omg.org/docs/ormsc/05-04-01.pdf> Accessed on September 7th 2005.

Ouadou, K. E. 1994. AMF : un modèle d'architecture multi-Agents Multi-Facettes pour interfaces homme-machine et les outils associés. Thèse de doctorat. École Centrale de Lyon. France.

Paternò, F. 2000. *Model-Based Design and Evaluation of Interactive Applications.* 208 pages. ISBN 1-85233-155-0. Springer.

Pfaff, G. 1985. User Interface Management System. Seeheim Workshop. Springer-Verlag. Berlin. Germany.

Ram, Janaki D., Raman K. N. Anantha and K. N. Guruprasad. 1997. A Pattern Oriented Technique for Software Design. ACM SIGSOFT. Software Engineering Notes. Vol. 22. No 4. Page 70.

Riehle, Dirk. 1997. Composite Design Patterns. OOPSLA 1997: 218-228.

Rising, L. 1996. Reuse at AG Communication Systems = Patterns. MultiUse Express. Vol. 4. No3.

Rossi, Gustavo, Fernando Lyardet and Daniel Schwabe. 1999a. Developing hypermedia applications with methods and patterns. ACM Computer. Survey. 31(4es): 8.

Rossi, Gustavo, Daniel Schwabe and Fernando Lyardet. 1999b. Designing Hypermedia Applications with Objects and Patterns. International Journal of Software Engineering and Knowledge Engineering. 9(6). pp 745-766.

Rossi, Gustavo, Daniel Schwabe and Fernando Lyardet. 1999c. Improving Web Information Systems with Navigational Patterns. Computer Networks 31(11-16). pp 1667-1678.

Schlungbaum, Edberg. 1996. Model-Based User Interface Software Tools – Current State of Declarative Models. Technical Report 96-30. Graphics, Visualization and Usability Center Georgia Institute of Technology. (Georgia). USA.

Schmidt Douglas C., Stal Michael, Rohnert Hans and Buschmann Frank. 2000. *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects.* Wiley & Sons. ISBN 0-471-60695-2.

Seffah, Ahmed and Gaffar, Ashraf. 2006. Model-based user interface engineering with design patterns. Journal of Systems and Software, doi:10.1016/j.jss.2006.10.037. 15 pages.

Shaw, Mary and David Garlan. 1996. *Software Architecture.* Prentice Hall, ISBN 0-13-182957-2.

Si Alhir, Sinan. 2003. Understanding the Model-Driven Architecture (MDA). Methods & Tools, Vol. 11. No.3. pp. 17-24. Online. <http://www.methodsandtools.com/PDF/Dmt0303.pdf>. Accessed on September 7th 2005.

Sinnig, Daniel. 2004. The complicity of patterns and Model-Based UI Development. Master of Computer Science, Montreal, Concordia University, 148 p.

Soley, Richard and the OMG Staff Strategy Group. 2000. Model-Driven Architecture. OMG Group. Online. <ftp://ftp.omg.org/pub/docs/omg/00-11-05.pdf>. Accessed on September 7th 2005.

Souchon, Nathalie, Quentin Limbourg and Jean Vanderdonckt. 2002. Task Modelling in Multiple Contexts of Use. In *Proceedings of DSV-IS 2002*. pp. 77-95. Rostock. Germany.

Sowa, J.F. and John A. Zachman. 1992. Extending and Formalizing the Framework for Information Systems Architecture. IBM Systems Journal. Vol. 31. No. 3. IBM Publication. G321-5488.

Sun Microsystems. 2001. *Java Look and Feel* Design Guidelines. Publisher Addison Wesley Professional. Online. <http://java.sun.com/products/jlf/ed2/book/>. Accessed on February 5th 2007.

Sun Microsystems. 2002a. Architecture multi-tiers. Online. <http://java.developpez.com/archi_multi-tiers.pdf>. Accessed on April 10th 2006.

Sun Microsystems. 2002b. Core J2EE Patterns Architecture. Online. <http://java.sun.com/blueprints/corej2eepatterns/Patterns/index.html >. Accessed on September 10th 2005.

Sun Microsystems. 2002c. J2EE Patterns Catalog. Online. <http://java.sun.com/blueprints/patterns/catalog.html>. Accessed on September 10th 2005.

Taleb, M., H. Javahery and A. Seffah. 2006. Pattern-Oriented Design Composition and Mapping for Cross-Platform Web Applications. The XIII International Workshop. DSVIS 2006. July 26-28 2006. Trinity College Dublin Ireland. DOI 10.1007/978-3-540-69554-7. ISBN 978-3-540-69553-0. Vol. 4323/2007. Publisher Springer-Verlag Berlin Heidelberg. Germany.

Taleb, M., A. Seffah and A. Abran. 2007a. Pattern-Oriented Architecture for Web Applications. 3rd International Conference on Web Information Systems and Technologies (WEBIST). March 3-6, 2007. ISBN 978-972-8865-78-8. pp. 117-121. Barcelona. Spain.

Taleb, M., A. Seffah and A. Abran. 2007b. Model-Driven Design Architecture for Web Applications. The 12th International Conference on Human Centered Interaction International (FIC-HCII). July 22-27, 2007. Beijing International Convention Center. Beijing. P.R. China. Vol. 4550/2007. pages 1198-1205. Publisher Springer-Verlag Berlin Heidelberg. Germany.

Taleb, M., A. Seffah and A. Abran. 2007c. Patterns-Oriented Design for Cross-Platform Web-based Information Systems. The 2007 IEEE International Conference on Information Reuse and Integration (IEEE IRI-07). August 13-15, 2007. pages 122-127. Las Vegas. USA.

Taleb, M., A. Seffah and D. Engelberg. 2007d. *From User Interface Usability to the Overall Usability of Interactive Systems: Adding Usability in System Architecture.* In A. Seffah, J. Gulliksen and M. Desmarais. (Eds). Human-Centered Software Engineering. Volume II. Software Engineering Models. Patterns and Architectures for HCI. Chapter 9. May 28th 2007. John Wiley & Sons. New York. USA.

Taleb, M.. A. Seffah and A. Abran. 2008a. On the suitability of XML and its quality framework of languages based on models for representing interactive systems. The 6th International Workshop on XML Technology Applications (XMLTech08). The 2008 World Congress in Computer Science. Computer Engineering and Applied Computing (WORLDCOMP'08). July 14-17, 2008. Las Vegas. Nevada. USA.

Taleb, M., A. Seffah and A. Abran. 2008b. Reconciling Usability and Interactive System Architecture Using Patterns. Journal of Software and Systems.

Taleb, M., A. Seffah and A. Abran. 2008c. Investigating Model-Driven Architecture for Web-based Interactive Systems. Journal of eMinds. (Submitted)

Taleb, M., A. Seffah and A. Abran, 2008d. POMA: A Pattern-Oriented and Model-Driven Architecture. Journal of Software - Practice and Experience. (Submitted)

Taleb, M. and A. Seffah. 2008e. Tracing the Evolution of Patterns as a Design Tool. Journal of Pattern Analysis and Applications. (Submitted)

Taleb M., A. Seffah and A. Abran. 2008e. Patterns + Personas = A Human-Centric Infrastructure for Web Applications Design. The International World Wide Web Conferences (WWW2009), April 20-24, 2009, Madrid, Spain. (Submitted)

Tidwell, J. Common Ground. 1997. A Pattern Language for Human-Computer Interface Design. Online. <http ://www.mit.edu/~jtidwell/common_ground.html>. Accessed on September 24th 2005.

UPA. 2001. Usability- a Winning Experience. Online. <http://www.usabilityprofessionals.org/conference/1997-2002overview.html>. Lake. Las Vegas: (Nevada). USA. Accessed on March 13th 2007.

Welie, M.V. 1999. 'The Amsterdam Collection of Patterns in User Interface Design. Online. <http://www.welie.com/patterns/>. Accessed on September 16th 2005.

Wills, Alan C. and Desmond D'Souza. 1996. Component and Framework-based Development. OOIS 1996. Pages 413.

Wirfs-Brock, Rebecca and Brian Wilkerson. 1989. Object-Oriented Design: A Responsibility-Driven Approach. OOPSLA 1989: 71-75.

Yacoub, Sherif and Hany Ammar. 2003. *Composition of Design Patterns.* Addison Wesley Professional. ISBN 0-201-77640-5. 416 pages. Germany.

Zachman, John A. 1987. A Framework for Information Systems Architecture. IBM Systems Journal. Vol. 26. No. 3. IBM Publication. G321-5298.

Zimmer, W. 1994. *Relationships Between Design Patterns.* In Patterns Languages of Program Design. Addison-Wesley.