Managing User Interface Pattern Collections

A thesis presented in partial fulfilment of the requirements
for the degree of
Master of Science in Computer Science
at Massey University, Palmerston North, New Zealand.

Junhua Deng

2006

Abstract

The research presented in this thesis describes the development of a comprehensive UI pattern management tool, MUIP, to support researchers and UI designers manipulate and explore a repository of UI pattern collections.

The concept of patterns originated from Alexander's pattern language for the architecture domain. Later, the software development and HCI communities adopted the pattern concept. Many disparate UI pattern collections have been developed and published using various media, such as books, internet, etc. Various pattern formats were used in these collections. In 2003, to cope with this problem, a group of HCI researchers developed a standardised pattern form, called PLML. Researchers have authored patterns, investigated the characteristics of pattern collections and also identified many of the functions required to manage pattern collections.

A framework for MUIP has been developed in the light of the analysis of the relevant literature and a survey of existing pattern tools. The framework supports the following features: pattern authoring, manipulating forces, browsing patterns, searching patterns, versioning and customising patterns, relating patterns, manipulating collections and importing or exporting patterns. Patterns are described using the standard pattern form (PLML). An enhanced version of PLML, called PLML v1.2, has been developed so that pattern contents can be organised more effectively.

Based on this framework, a specification of a comprehensive pattern management system for manipulating pattern collections was developed and a prototype implemented accordingly. A formal evaluation confirmed the usefulness of the prototype.

Acknowledgements

First of all, I want to thank my supervisor, Associate Professor Elizabeth Kemp. She introduced me to the research area of UI patterns and tools. I am especially thankful to her conscientious and patience in the research and for her helpful insights and guidance.

Thanks also go to my second supervisor, Ms Lis Todd for her co-supervision, and for her helpful suggestions in regard to the research.

In addition, I would like to thank Mr. Robert Thompson for proofreading and suggesting corrections to the thesis.

Finally, I would give many thanks to my parents, my wife Yu Liu, my sisters and my brother for all their support and love during these years.

Related Publication

Deng, J., Kemp, E., & Todd, E. G. (2005). Managing UI pattern collections. In *Proceedings of the 6th ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction: Making CHI Natural* (Auckland, New Zealand, July 07 - 08, 2005). CHINZ '05, vol. 94. ACM Press, New York, NY, 31-38.

List of Contents

List of Figures

List of Tables

Chapter 1 Introduction

1.1 Background

From ordinary life, people or organizations have recognised and defined various rules, guidelines and standards to support achieving a good solution for a real design problem. Following these principles, designers can facilitate the development process and control the quality of the design to achieve the proposed design goal easily. For example, a group of builders can construct a house of good quality based on predefined templates in a reasonably short period (Alexander *et al.*, 1977).

Templates emerge from recognised principles of good design. They form, as it were, a pattern for others to follow. The concept of a "pattern" was first proposed by the architect, Christopher Alexander, in his PhD thesis which was subsequently published as the book *Notes on the Synthesis of Form* (1964). In a later book, *A Pattern Language* (Alexander *et al.*, 1977), the authors provide a definition of a pattern and introduce a detailed collection of patterns as a language in the urban architectural domain. A pattern is not invented, but it is identified from the fundamental principles of a good design. A pattern is a narrative description, which represents a design problem and a solution for the design problem within a specific context. A pattern not only points out an invariant solution to a problem with various goals and constraints, but also explains the relationship between them and how the pattern balances the constraints. The idea of architectural patterns is to capture knowledge of good architectural design and document this knowledge in a way that can be used as common vocabularies by the community. The aim of Alexander's patterns is to help people build a town or a building which has "quality without a name" (Alexander *et al.*, 1979). More details about Alexander's patterns will be introduced in Section 2.1.

In 1987, the software engineering community adopted the pattern concept from the architectural domain. Beck & Cunningham (1987) outlined a pattern language in object-oriented design and summarised five patterns for user interface design at the OOPSLA-87 workshop in Orlando. The authors described an experiment where novice designers of Smalltalk succeeded in designing their own Smalltalk user interface after mastering some basic Smalltalk user interface concepts from those five patterns. The initial success in

interface design motivated the authors to write a pattern language for object-oriented design. The OOPSLA workshops and software engineering community continued to discuss and exchange software patterns after 1987. More details about software patterns will be described in Section 2.2.

Mode Cursor

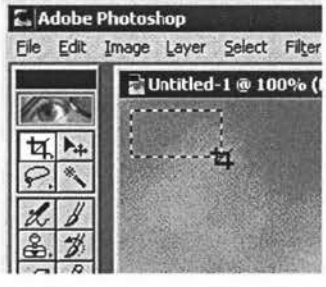
Author	Martijn van Welie
Problem	The user is creating or modifying an object and needs to know which edit function is selected.
Principle	Immediate Feedback (Feedback)
Context	In many direct manipulation applications the users first selects a tool/function, thus entering a special mode/state, and then works on an object. Since such applications usually offer many functions to create or modify objects.
Forces	• Not every function may have an icon or shape. • Completing a function may cause several intermediate states which may also need to be shown. • The user needs immediate feedback on which function was selected, i.e. which mode/state the system is in.
Solution	**Show the interface state in the cursor.** The interface state changes many times during interaction, for instance when a function is selected or when an action such as dragging is performed. Therefore, show the current state to the user by changing the cursor. The cursor can be changed to an icon or some other shape that gives feedback about the current interface state. Change the cursor back to a neutral cursor if the function is completed or deactivated.
Rationale	The cursor gives extra feedback about the active function. The user watches the cursor when performing a function so it is the most appropriate place on the screen to give feedback i.e. the user does not need to look at another portion of the screen. The solution increases satisfaction and may decrease errors.
Examples	

Figure 1.1 Mode Cursor pattern from Welie's pattern collection (2004a).

In 1986, the earliest related reference in the human computer interface (HCI) literature to the Alexandrian pattern concept appears in Norman and Draper's book, *User Centered System Design* (1986). However, the HCI community did not pay much attention to patterns until the workshop "Putting it All Together: Pattern Languages for Interaction Design" at CHI'97, much later than the software engineering community. Since then, different researchers have developed many UI patterns and pattern collections. Some pattern collections are disseminated in related workshops (Welie *et al.*, 2002) whilst others are published on the web (Tidwell, 1999; Welie, 2004b; Tidwell, 2002; Graham, 2003b; Laakso, 2003) or in

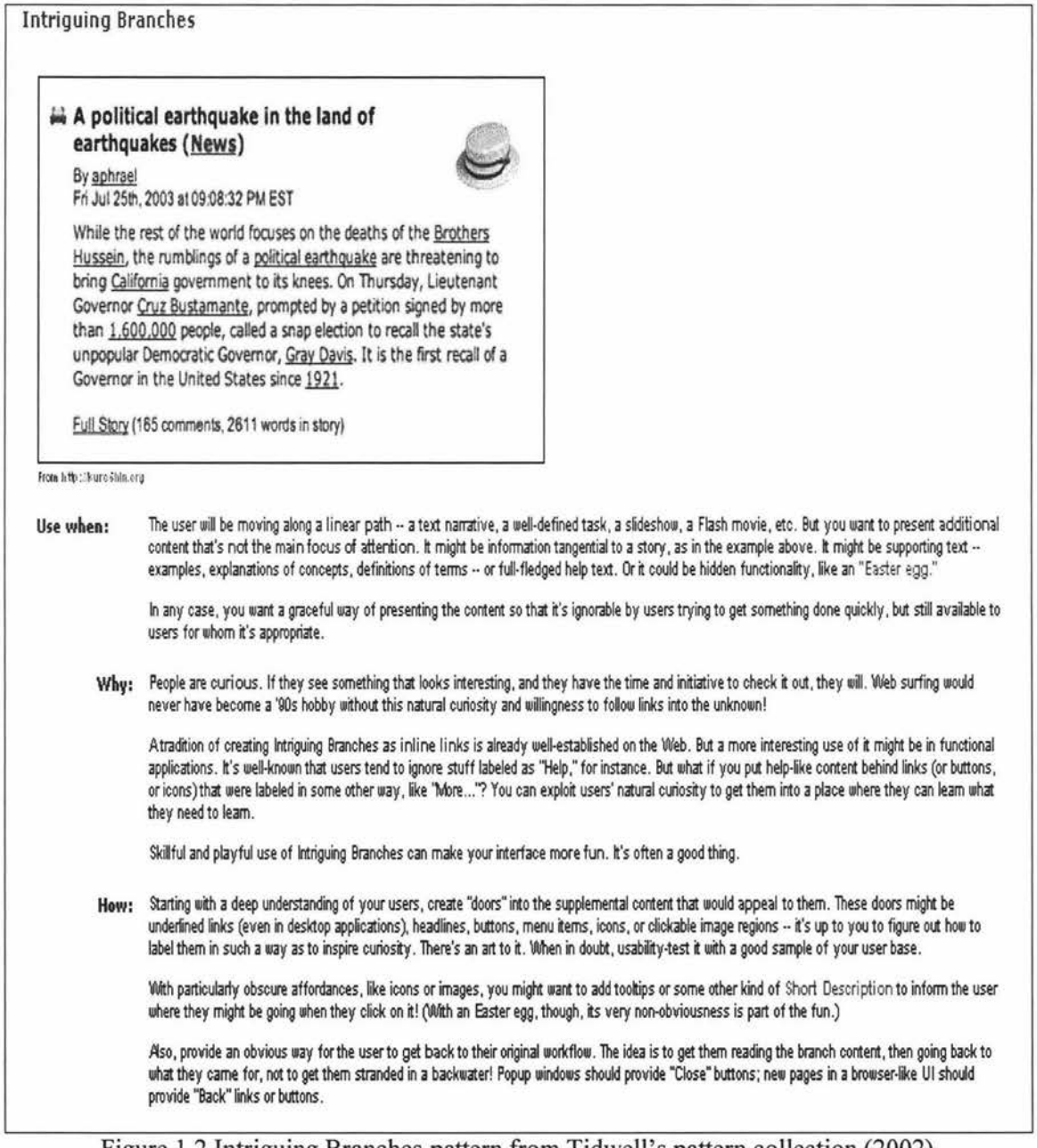
books (Borchers, 2001; Duyne *et al.*, 2003; Graham, 2003a). Although the pattern concept originates in Alexander's work (1964), pattern authors prefer to use their own forms when writing UI patterns. As a result, there exist many variant pattern forms among these collections. Figure 1.1 and Figure 1.2 illustrate two different patterns from Welie's and Tidwell's pattern collections. These two patterns are written in distinct pattern forms. In Chapter 2 various major pattern forms will be analysed.

A collection of patterns might be seen as a pattern language. Todd *et al.* (2004) investigated the factors which determine whether a collection could be considered a pattern language. Some elaborate collections cannot as yet be considered a pattern language as they do not meet the criteria. For this reason, the generic word "collection" is used to refer to a set of patterns in this thesis, even when the set of patterns could be classed as a pattern language.

Borchers (2001) has claimed that there is a high level of agreement on the need to construct a pattern language. The author believes that there will be a common form for HCI patterns, which derives from current various HCI pattern forms and unifies them. Welie *et al.* (2000) first used XML (Extensible Markup Language) to create a standard and consistent pattern format for publishing HCI patterns in their pattern catalog website. Later, some researchers advised the use of XML to document patterns for pattern tools (Borchers, 2001; Greene *et al.*, 2002). The CHI 2003 Pattern Workshop produced a significant outcome for the structure of HCI patterns in the specification of the Pattern Language Markup Language (PLML) (Fincher, 2003). PLML uses XML to document pattern data content in a consistent form based on the standardised XML DTD or Schemas, which can be found in Appendix A. PLML can be applied to unify the format of existing HCI pattern collections. The details of PLML are described in Chapter 2.

The HCI community applies patterns in solving user interface design tasks. This raises the issue of the creation, management and use of pattern collections. Computer systems can facilitate the activities of pattern composition and consumption. The potential users of such a computer system fall into four categories according to the main tasks they may wish to perform in relation to pattern collections: end users, UI researchers, UI designers, and software developers. The intuitive examples of pattern collections can enhance the knowledge of the end users and aid them to express their requirements to the designers. The UI pattern collections act as the common language to facilitate the communication between end users and designers. For UI researchers, or UI designers, who wish to build their own

patterns or collections based on the work of others, it is necessary to have a suitable tool to access and manipulate the available pattern collections. The knowledge of UI pattern collections enables the software developer to review whether the UI patterns have been applied in the software project properly. This research will focus on the needs of the UI researchers and designers.


Intriguing Branches

A political earthquake in the land of earthquakes (News)

By aphrael
Fri Jul 25th, 2003 at 09:08:32 PM EST

While the rest of the world focuses on the deaths of the Brothers Hussein, the rumblings of a political earthquake are threatening to bring California government to its knees. On Thursday, Lieutenant Governor Cruz Bustamante, prompted by a petition signed by more than 1,600,000 people, called a snap election to recall the state's unpopular Democratic Governor, Gray Davis. It is the first recall of a Governor in the United States since 1921.

Full Story (165 comments, 2611 words in story)

From http://kuro5hin.org

Use when: The user will be moving along a linear path -- a text narrative, a well-defined task, a slideshow, a Flash movie, etc. But you want to present additional content that's not the main focus of attention. It might be information tangential to a story, as in the example above. It might be supporting text -- examples, explanations of concepts, definitions of terms -- or full-fledged help text. Or it could be hidden functionality, like an "Easter egg."

In any case, you want a graceful way of presenting the content so that it's ignorable by users trying to get something done quickly, but still available to users for whom it's appropriate.

Why: People are curious. If they see something that looks interesting, and they have the time and initiative to check it out, they will. Web surfing would never have become a '90s hobby without this natural curiosity and willingness to follow links into the unknown!

A tradition of creating Intriguing Branches as inline links is already well-established on the Web. But a more interesting use of it might be in functional applications. It's well-known that users tend to ignore stuff labeled as "Help," for instance. But what if you put help-like content behind links (or buttons, or icons) that were labeled in some other way, like "More..."? You can exploit users' natural curiosity to get them into a place where they can learn what they need to learn.

Skillful and playful use of Intriguing Branches can make your interface more fun. It's often a good thing.

How: Starting with a deep understanding of your users, create "doors" into the supplemental content that would appeal to them. These doors might be underlined links (even in desktop applications), headlines, buttons, menu items, icons, or clickable image regions -- it's up to you to figure out how to label them in such a way as to inspire curiosity. There's an art to it. When in doubt, usability-test it with a good sample of your user base.

With particularly obscure affordances, like icons or images, you might want to add tooltips or some other kind of Short Description to inform the user where they might be going when they click on it! (With an Easter egg, though, its very non-obviousness is part of the fun.)

Also, provide an obvious way for the user to get back to their original workflow. The idea is to get them reading the branch content, then going back to what they came for, not to get them stranded in a backwater! Popup windows should provide "Close" buttons; new pages in a browser-like UI should provide "Back" links or buttons.

Figure 1.2 Intriguing Branches pattern from Tidwell's pattern collection (2002).

The CHI 2003 Pattern Workshop (Perspectives on HCI Patterns: Concepts and Tools) is a milestone for HCI pattern research. The workshop had two main goals. The first goal related

to conceptual issues - to identify and share a variety of perspectives on patterns and pattern languages for design activities such as interaction design and HCI design. It is necessary to consider why and how patterns are created, and how and when they can be reused during the design process. For this goal, the workshop aimed (Fincher *et al.*, 2003, p.1045):

- *To identify what is important in the area, what is fundamental to patterns as applied to HCI;*
- *To enhance our understanding of the concept of patterns and pattern languages in relation to other theoretical models used in design;*
- *To work towards the creation of a "map" of the conceptual territory of the pattern endeavour.*

The second goal related to computer tools - to understand how computer systems can efficiently support a variety of activities during building and reusing the patterns and related software components. For this goal, the workshop aimed (Fincher *et al.*, 2003, p.1045):

- *To create a shared understanding of the needs and activities of the diverse groups of people involved in pattern creation and pattern use;*
- *To identify ways in which the capabilities of the computer can enhance and support these activities.*

The research described in this thesis is concerned with identifying the need and activities of UI researchers and designers involved in pattern authoring and pattern consumption, and then conceptualising a pattern management tool to support these needs and activities.

1.2 Motivation and Objectives

The motivation for this research comes from three perspectives. First, for researchers and UI designers who wish to build their patterns or collections based on the work of others, it is necessary to have a suitable pattern tool to access and manipulate the available pattern collections. Ideally, such a tool will be based on a structured pattern structure, such as that specified by PLML. Second, none of the existing UI pattern tools have totally supported PLML to provide a standardised format for pattern authoring and consumption. Last, because the specification of PLML made in CHI 2003 Pattern Workshop is still in the early stages, developing a UI pattern tool, which totally supports PLML, will help evaluate the specification.

Computer systems can facilitate the activities of pattern composition and consumption, and therefore, have been proposed to support the creation and use of patterns or collections. A pattern tool could support the role-specific activities to manage patterns and collections, and the different tasks required by various user groups such as HCI design practitioners, software engineers, and domain experts (Borchers, 2001; Greene *et al.*, 2002; Gaffar *et al.*, 2003; Schuemmer, 2003b).

Although most of the existing pattern tools outline the importance of using a standardised format for patterns, none have totally implemented PLML as yet. Also, since the PLML specification is still in its initial version, there is a need to develop a prototype in order to evaluate the PLML specification. Development of an effective pattern management tool with PLML could greatly contribute to the UI pattern domain.

Based on the above motivations, we need to consider the research question: how can a tool be developed to support the management and use of the collection of UI patterns based on PLML?

The objectives of this research are to investigate the functions required for both the creation and manipulation of pattern collections within a pattern management tool based on PLML and how these can be best supported.

The research aims:

- To identify from the literature the needs and activities of those developing pattern collections;
- To analyse current tools to determine to what extent they meet the needs of those using pattern collections;
- To conceptualise a pattern management tool to manipulate pattern collections;
- To design, implement and evaluate a prototype to support the conceptualisation of a tool;
- To evaluate the specification of PLML.

Related knowledge from human computer interaction and database management is used to conceptualise and implement a pattern management system to manipulate and use UI pattern collections.

1.3 Research Approach

The research approach employed to achieve the objectives described above is illustrated in Figure 1.3 and summarised as follows:

- The research approach focused on the research problem: how to develop a tool to support the management and use of collections of UI patterns based on PLML?
- Based on the research problem and the motivation described in Section 1.2, the literature review, investigates the needs and activities for such a tool. To identify the requirements of the tool, existing specifications were investigated. A number of tools supporting the management of patterns were also evaluated.
- Based on the literature review and analysis of existing tools, a specification for a tool to manage pattern collections has been defined.
- A prototype has been implemented based on the requirements.
- The evaluation phase validates the features of the prototype. Overall, this is an iterative process of specification, prototyping and evaluation.
- A discussion considers to what extent the implementation of the prototype meets the objectives of this research.

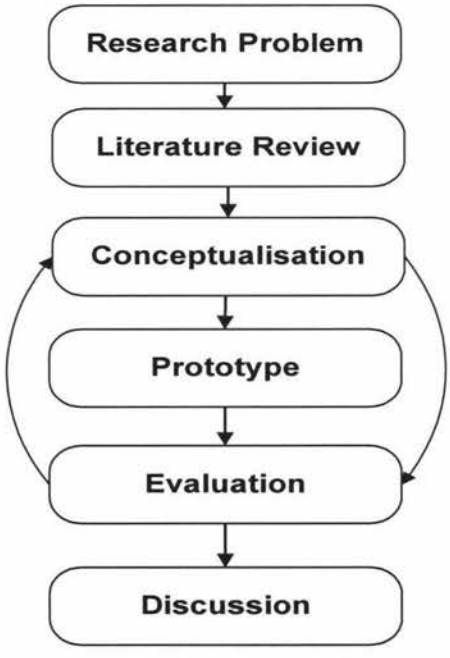


Figure 1.3 Steps in the research approach.

1.4 Outline of Thesis

This thesis consists of six chapters. Chapter 1 is the introductory chapter. Chapter 2 reviews the background of HCI patterns, and describes the types of relationships between patterns and PLML in detail, as well as introducing the categorisation of patterns. Chapter 3 reviews the existing specifications and tools for managing pattern collections. The conceptualisation of a new UI pattern management system is identified. Chapter 4 describes the prototype design of a pattern management tool (MUIP). A revised version of PLML is presented within this chapter. The detailed prototype for managing pattern collections with related implementation issues is also outlined in this chapter. Chapter 5 discusses the evaluation of the project. Finally, Chapter 6 gives an overall conclusion and a description of future work.

Chapter 2 Literature Review

This chapter provides a detailed literature review. The origin of the pattern concept and the appearance of patterns in software engineering and the HCI domain will be described. As the goal is to develop an HCI pattern tool, more emphasis will be put on the literature review of the HCI patterns. Patterns and guidelines will be compared, as well as the different types of relationships between patterns proposed by researchers. Various pattern forms, including PLML will be reviewed, and the forces and categorisation of patterns will also be discussed.

2.1 Alexander's Pattern Concept

As mentioned in Chapter 1, a pattern is a description of a design problem and a solution within a specific context. Alexander (1979) described it in more detail:

> *Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution...each pattern is a relationship between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain spatial configuration which allows these forces to resolve themselves. ... The pattern is, in short, at the same time, a thing which happens in the world, and the rule which tells us how to create that thing, and when we must create it (p.247).*

The above definition shows that a pattern can be considered as a three-part rule, which not only captures a solution to a problem with various goals and constraints, but also explains the relationship between them and how the pattern balances the constraints.

The idea of architectural patterns was to capture knowledge of good design and document it in a way that can be shared by the community in this domain. The aim of Alexander's patterns was therefore to help people build a town or a building with the "quality without a name" (Alexander, 1979). Alexander's patterns are written in a pattern form where the components are in the same order, as listed in Table 2.1. Note that *picture* shows an archetype of the pattern, and *diagram* shows a generalised sketch of how it is implemented.

Although each of Alexander's patterns contains the above components, there is no label for every part of the pattern. For example, there are no headings such as "Problem" or "Solution". However, all patterns in the pattern language follow typographical rules (Alexander *et al.*, 1977; Borchers, 2001):

- Each pattern has the same components.
- Each part of a pattern is present in the same printing style.
- Each part of a pattern can be further distinguished by specific signs or words.

Table 2.1 Alexandrian pattern form

Component	Description
Number	The number ranges from 1 to 253.
Name	The name captures pattern information in a few words and makes it easy to remember and refer to.
Ranking	A pattern is ranked by being marked with 0, 1 or 2 asterisks, which indicate how much confidence the authors had in the pattern.
Picture	A picture illustrates the situation where the pattern applies.
Context	Context explains the pre-condition or environment when the pattern will be applied.
Problem Statement	A short statement summarises the essence that the pattern addresses.
Problem Description	A detailed description of the problem describes the objectives and goals with forces and examples.
Solution	The solution contains instructive rules to explain how to solve the problem and realise the desired outcome.
Diagram	The visualisation of the solution is a diagram that sketches the main idea graphically.
References	The references are used to guide readers to the smaller-scale patterns within the pattern language.

Alexander and his co-authors argue that there is no isolated pattern. Therefore, patterns in the collection refer to others that are arranged in hierarchies based on a scale. The smaller-scale patterns align at the lowest level of the hierarchy and larger-scale patterns align at the highest level of the hierarchy. The relationships between patterns are used to combine patterns together to form a hierarchical pattern language for a specific architectural design (Borchers, 2001). In other words, Alexander *et al* (1977) describe their patterns as components in a pattern language. Each larger-scale pattern includes smaller-scale sub-patterns. Therefore, individual patterns always refer to related patterns. The architecture pattern language includes 253 patterns that are ordered from large-scale patterns to smaller-scale patterns. In another book, *The Timeless Way of Building*, Alexander (1979) explains the discipline required to use these patterns when constructing a building or town. The authors use the example of the University of Oregon to explain how the pattern language can be applied in practical architecture design (Alexander *et al.*, 1975). Mahemoff and Johnston (1998b) observed that the main contribution of a pattern language is that designers can start from a specific context and navigate all related patterns to produce a design.

2.2 Patterns in Software Engineering

The pattern concept from the architectural domain was brought into the software engineering community. Pattern Definition Thread (Gabriel, n.d., para.9) is a concise software pattern definition based on the Alexandrian three-part rule: "Each pattern is a three-part rule, which expresses a relation between a certain context, a certain system of forces which occurs repeatedly in that context, and a certain software configuration which allows these forces to resolve themselves." Gamma *et al* (1995, p.3) provide a specific definition for software design patterns: "The design patterns are descriptions of communication objects and classes that are customised to solve a general design problem in a particular context".

Both the above software pattern definitions include the essential components of the Alexandrian pattern concept, which describe the relationships between the design problem, context and solution. Gabriel's definition is the more general one. It outlines the component of "forces" required to be resolved, which is very similar to the Alexandrian description. The other one focuses on the objected-oriented software design pattern.

In 1995, the "Gang of Four" (GOF) collected and published a collection of object-oriented software design patterns in their book, which made the software pattern concept widely accepted by the software engineering community (Gamma *et al.*, 1995). In this book, 23 software design patterns were gathered in a collection, not a pattern language. Although some patterns link to others in the collection, the network is not complete enough to form a software pattern language. The patterns are classified into three categories: creational patterns, structural patterns and behavioural patterns.

The pattern structure used in Gamma *et al*'s book contained the main components of Alexander *et al*'s pattern format. The main content includes the name, context, problem, solution, examples, diagrams and references (Borchers, 2001). But the pattern form has different headings for some of those main constituents. Gamma *et al*'s pattern form (1995) is described in Table 2.2.

Many researchers have explored how to apply the pattern concept to software engineering domain. For example, the annual Pattern Languages of Programming (PLoP) started in 1994 (Coplien & Schmidt, 1995). This conference provides opportunities for researchers to discuss and exchange new software design patterns. In the first PLoP conference, Coplien

and Schmidt (1995) illustrated the generative development-process pattern language in a specific form, which was named as the Coplien (1996).

Table 2.2 GOF pattern form

Component	Description
Pattern Name/ Classification	Pattern name is used to refer to the pattern and classification, which puts the pattern into different categories.
Intent	A short description explains the rationale and objective of the pattern.
Also Known As	It is the alternative well-known name for the pattern.
Motivation	A scenario illustrates the design problem.
Applicability	The applicability describes the application context of the pattern.
Structure	The structure illustrates objects and their relationships in graphical representation.
Participants	It claims the candidate objects/classes and their responsibilities.
Collaborations	A description explains how the participants carry out their responsibilities.
Consequences	It presents the trade-offs and the results of using the pattern.
Implementation	It describes the related issues regarding implementing the pattern.
Sample Code	Pieces of code explain how to implement the pattern in a specific programming language.
Known Uses	It refers to existing examples in real systems.
Related Patterns	This section describes other patterns related to the pattern.

This pattern form also contains the basic components from the Alexandrian pattern form. The differences are that:

(i) The Coplien pattern form contains a component separately called "force", which presents the design constraints and trade-offs;

(ii) The Coplien pattern form contains a component called "rationale", which describes the principles of the pattern.

Although both these software pattern forms reflect the main components of the Alexandrian pattern form, Coplien's pattern form seems to be more similar to the Alexandrian pattern structure. However, both Gamma *et al*'s pattern form and Coplien's adopt headings to separate the components inside a pattern, which extends the Alexandrian typographical rules. This method is widely accepted by both the software engineering community and the HCI community.

Software design patterns act as a useful common language for the community to discuss and share ideas regarding design problems. These patterns record good solutions to specific design problems and new designers can learn and apply them in practical design immediately. Therefore, software design patterns are created for designers, not for the end users of the software.

2.3 Patterns in Human Computer Interaction (HCI)

Alexander's pattern concept was introduced to the HCI domain at the OOPSLA conference in 1987 (Beck & Cunningham, 1987). The three-part rule for patterns was also adopted by HCI researchers. Tidwell (1999) explains the concept of a pattern as follows: "A pattern describes possible good solutions to a common design problem with a certain context, by describing the invariant qualities of all those solutions" (Preface). Tidwell's pattern definition emphasises that the design problem could be a common and recurring design problem. The solutions for the design problem are required to be of good quality, which is consistent with the Alexandrian "quality without a name" (after p.16).

Recently, many HCI researchers have explored why and how to apply pattern knowledge in the HCI area. Both Erickson (2000) and Tidwell (1999) point out the reason for requiring patterns in HCI design is that they provide a *lingua franca* (common language). Other experts have also investigated the benefits of a pattern language in HCI. Appleton (1997, Pattern languages section, para.1) describes a pattern language as follows: "a pattern language is a collective of such solutions which, at every level of scale, work together to resolve a complex problem into an orderly solution according to a pre-defined goal".

The above definition outlines that patterns are arranged in a hierarchy for a pattern language based on a scale factor. To achieve the design goal, the patterns link and cooperate with each other. Salingaros (2000) also points out the "hierarchical connections across scales" and the relationships between patterns. He emphasised that connectivity rules cause a coherent combination of patterns to construct a pattern language: "A pattern is an encapsulation of forces; a general solution to a problem. The 'language' combines the nodes together into an organisational framework. A loose collection of patterns is not a system, because it lacks connections" (p.154). The connections between patterns are also highlighted by Pemberton (2000) who says:

> *[A pattern language] enables the collection of patterns to operate generatively, each pattern showing the sub-patterns required to resolve more detailed design issues, in the context of the larger design (Section 3).*

A formal definition for a pattern language was introduced by Borchers (2001), which highlights the connectivity required to make a collection of patterns into a pattern language. Borchers describes the notation of the model as follows (p.52):

1. *A pattern language is a directed acyclic graph PL = (φ,$\mathcal{R}$) with nodes $\varphi=\{P_1,...P_n\}$ and edges $\mathcal{R}=\{R_1,...R_m\}$.*
2. *Each node P$\in\varphi$ represents a pattern.*
3. *For two nodes P, Q $\in\varphi$, we say that P references Q if, and only if, there is a directed edge R $\in\mathcal{R}$ leading from P to Q.*
4. *The set of edges pointing away from a node P $\in\varphi$ is called its references, and the set of edges pointing to it is called its context.*
5. *Each node P $\in\varphi$ is itself a set P= {n, r, i, p, $f_1...f_i$, $e_1...e_j$, s, d,} of a name n, ranking r, illustration i, problem p with forces $f_1...f_i$, examples $e_1...e_j$, the solution s, and diagram d.*

All of the above different definitions reflect the essential principle of a pattern language. Todd *et al.* (2004) summarise the principle as: "Grouping that links lower-level patterns into higher-level patterns creates a hierarchy of scale". Thus, a user interface can be presented at different hierarchical levels within a pattern language. Todd *et al.* also investigate the factors determining whether a collection can be considered a pattern language. Therefore, a collection of patterns might be a pattern language. On the other hand, some insufficiently developed collections cannot, as yet, be considered pattern languages until they are developed completely. For this reason, the generic word "collection" is used to present a set of patterns in this thesis, even when the set of patterns can be classed as a pattern language.

There are many techniques to help designers implement a high quality interface design, such as guidelines, standards, and heuristics. One question arises here: why should the HCI community adopt the pattern concept and try to develop a pattern language for user interface design while there are many sets of guidelines? The issues raised by this question are discussed in the following section.

2.3.1 Patterns and Guidelines

Guidelines that document user interface design knowledge and experience have been formulated in order to guide users to design good quality user interfaces (Smith & Mosier, 1986; Brown, 1988; Mayhew, 1997). From Tidwell's pattern definition, patterns and guidelines seem to have a similar purpose even though they are written in different forms. In the following paragraphs, a comparison will be made between patterns and guidelines.

Patterns are well-structured templates with rich information about the context of a problem. They are easy to read, allowing a user to focus on specific fields. Users can quickly find and identify a target pattern based on context and related fields. Guidelines are short, simplistic, and abstract so that they are hard to interpret and select (Mahemoff & Johnston, 1998b; Welie *et al.*, 2000; Griffiths & Pemberton, 2001). Although many guidelines contain the context for the design problem, the context is usually unclear so it becomes difficult to identify a guideline for a specific context (Mahemoff & Johnston, 1998b). Guidelines could be written using a pattern template, but this method cannot guarantee that the guidelines will be a pattern (Welie *et al.*, 2000). Some efforts have been made to create patterns from a number of guidelines (Skogseid & Spring, 1995). As patterns contain more design knowledge than guidelines, sometimes individual patterns can be composed of several guidelines (Welie *et al.*, 2000).

Patterns include a rationale section to explain why the pattern can work and the users will know the pattern contains a good solution to the problem. However, individual guidelines usually do not contain an explanation of the associated rationale (Welie *et al.*, 2000; Dix *et al.*, 2004). Users have to discover the rationale by themselves.

Patterns explicitly express both the problem and the context of use within the pattern description. A proven solution is provided by each pattern (Welie *et al.*, 2000), while the validity of guidelines is based on the user understanding the context of use and applying the guidelines properly (Welie *et al.*, 2000). However, the problem and context for use within guidelines is not well defined and is abstract. Therefore, it is difficult to ensure users apply guidelines to solve the design problem adequately.

Patterns identify a number of examples from real life that show good use of the pattern. The examples and also references in the solution section provide evidence as to the quality of the pattern. These references give the designer confidence that the solution they create using the pattern will probably be successful. Guidelines rarely provide examples or evidence of successful application. It is usually the authority of the author that gives guidelines their validity.

Related patterns can form a pattern language to develop a complete design (Welie, 2001; Dix *et al.*, 2004). A pattern can refer to relevant patterns, which are used to solve sub-problems. By this method, related design knowledge can construct a semantic network and make it easy

for users to browse the knowledge related to a specific design problem. However, it is impossible for guidelines to do this.

Patterns can act as a communication medium among all stakeholders (Griffiths & Pemberton, 2001; Dix et *al.*, 2004). Patterns are written in an intuitive way with high readability. Therefore, interaction design can involve users, HCI designers and software engineers. Guidelines are very abstract and difficult to understand. Usually, guidelines are created for designers.

Table 2.3 Comparison between guidelines & patterns

Features / Name	Patterns	Guidelines
Content	Well-structured/rich/concrete	Simplistic/abstract
Explanation of rationale	Always	Sometimes
Degree of validity	High	Low
Explicit degree of problem, context	High	Low
Difficult to select	Sometimes	Often
Difficult to interpret when used	No	Yes
Completeness of design	Complete	Partially
Communication medium among all stakeholders	Yes	No
Conflicting solutions	No	Yes
Explaining different levels of design knowledge	Yes	No

Guidelines can conflict with each other (Welie *et al.*, 2000) therefore alternative conflicting solutions can be created even when created by the same designer. Patterns can be used to develop substitute solutions which are less likely to conflict because the forces defined in the problem description enable the designer to make informed choices when constructing alternatives. On the other hand when applying patterns, different designers may generate quite different solutions for the same problem because of the choices each designer makes.

Patterns at different levels within a hierarchy can be used to describe different levels of design knowledge. This design knowledge may be related to social and organisational topics which deal with different levels of design from conceptual design to concrete design (Dix *et al.*, 2004). Guidelines do not represent different levels of design knowledge. Table 2.3 summarises the above discussion.

Due to the advantages of patterns and the limitations of guidelines, some researchers prefer to write patterns rather than to write guidelines (Griffiths & Pemberton, 2001).

2.3.2 Pattern Collections

In recent years, many HCI patterns and pattern languages or pattern collections have been developed by different researchers. Some pattern collections are created and disseminated in workshops, such as by Welie *et al.* (2002). Some pattern collections are published on the Internet as open resources shared by the HCI community (Tidwell, 1999; Tidwell, 2002; Graham, 2003b; Laakso, 2003; Welie, 2004b), some pattern collections or pattern languages are published in books (Borchers, 2001; Duyne *et al.*, 2003; Graham, 2003a).

Researchers need to identify and understand the relationships between patterns or pattern collections for building up the entire knowledge of UI patterns while UI designers focus on getting guidance from the patterns to produce good interface solutions. A computer system could support these activities. The next two chapters provide a detailed discussion about how pattern tools can aid researchers or UI designers to reach their goals. The following section summarises the current discussion on the types of relationships between patterns.

2.3.3 Types of Relationships

Individual patterns do not stand-alone and the connectivity between patterns plays an important role in helping to achieve the design goal and objective for a complex UI design problem. However, the relationship between patterns is often complicated. A mechanism is required to explore such relationships and discover the types of links that occur.

Patterns can be combined in a variety of ways to explore solutions (Salingaros, 2000). The combination of patterns requires an understanding of the connectivity between patterns and the relationships that can facilitate the integration process (Noble, 1998). The relationships can assist the designers to identify the appropriate patterns to solve a specific design problem (Noble, 1998). Thus one research challenge is to identify the types of link or relationship between patterns. A number of researchers have explored the area and identified a number of types of relationships that occur between patterns.

Noble (1998) summarised and classified relationships between object-oriented design patterns. The relationships are classified into two groups (see Table 2.4): primary relationships and secondary relationships. Noble describes the secondary relationships in

terms of the primary relationships. For example, "*used by*" is the inverse of the "uses" relationship and "*similar*" is a less well defined version of the "*conflicts*" relationship. Although these relationships are identified from the software pattern domain, they probably can be used to express the relationships of patterns in other domains. At first glance it appears that this is already the case, for example, the "*uses*" relationship also links patterns in the architectural domain (Alexander *et al.*, 1977). Table 2.7 indicates that "*uses*" is employed in the HCI domain as well, but care has to be taken when interpreting same named relationships in different domains as they can have subtly different meanings and the relationships are not always clearly defined.

Table 2.4 Summary of the relationship types of OO design patterns

Primary Relationships	
uses	One pattern uses another pattern.
refines	A specific pattern refines a general pattern.
conflicts	A pattern addresses the same problem as another pattern.
Secondary Relationships	
used by	A smaller pattern is used by a larger pattern.
refined by	A general pattern is refined by a specific pattern.
variant	A variant pattern refines a more well-known pattern.
variant uses	A variant of one pattern uses another pattern.
similar	A pattern is similar to another pattern.
combines	Two patterns combine to solve a single problem.
requires	A pattern requires the solution of another problem.
tiling	A pattern uses itself.
sequence of elaboration	A sequence of patterns from the simple to the complex.

Although "*uses*" is the only relationship defined explicitly in Alexander et al's pattern language, an analysis of these links by Salingarios (2000) resulted in the description of five different types of coupling between patterns (see Table 2.5):

Table 2.5 Salingarios' relationship types (after p.150-151)

Proposed Relationships	**Description**
contains	One pattern contains another smaller-scale pattern.
complements	Two patterns are complementary and one needs the other for completeness.
overlaps	Two patterns solve distinct problems that overlap and coexist on the same level.
alternatives	Two patterns solve the same pattern in alternative, but equally valid ways.
derives	Distinct patterns share a similar structure, thus implying a higher-level connection.

Both Mullet (2002) and Welie *et al.* (2003) identified three basic relationship types in their respective collections (see Table 2.6). These relationships are based on the relationship types used when modelling objected-oriented classes. The derivation or "*is-a*" relationship and the aggregation or "*has-a*" relationship have similar meanings but the meaning of the

"*association*" relation is different with Welie's being more general than Mullet's as it includes alternatives.

Table 2.6 Mullet and Welie's relationship types

Relationships	Description
is-a	Distinct patterns share a similar structure, thus implying a higher-level connection.
has-a	One pattern contains another smaller-scale pattern.
uses or related-to	One pattern knows about another pattern. It cross-references that pattern because it could "use" the pattern (Mullet).
	A pattern can be "related to" other patterns because it occurs in the larger context of the design problem or the patterns are alternatives for the same problem (Welie).

Shuemmer (2003b) identifies seven relationship types between patterns and three relationship types between patterns and other artefacts (see Table 2.7). These relationships are not named and are not fully explained. Therefore, proposed names are used in Table 2.7.

Table 2.7 Shuemmer's relationship types

Proposed Relationships	Description (Between patterns)
uses	One pattern uses another pattern in its solution.
variant	A pattern is a variant of another pattern.
similar	A pattern has a similar problem as another pattern.
relates	A pattern is related in the related section to another pattern.
is-a	A pattern specialises another.
sequence	A pattern connects to another pattern as part of the sequence.
used-by	A pattern mentions another pattern in its context.
	(Between patterns and other artefacts)
same category	Two patterns are members of the same class or family.
same participant	Two patterns involve a common participant.
same known example	Two patterns are found in the same known use.

The context section of Alexander et al's patterns (1997) is normally just a list of patterns that may use that pattern when instantiated. Graham (2003a) defines this relationship type more succinctly as: "an arrow from pattern p1 to pattern p2 is to be interpreted as meaning 'p1 possibly generates a context for applying p2 and indicates that the designer should be applying p2 whenever she has applied p1" (p.53).

At the CHI 2003 pattern workshop (Fincher, 2003), only three types of pattern relationships were agreed upon when describing PLML (see Table 2.8). The "*is-a*" relationship defined for PLML is not the inheritance type relationship associated with modelling classes which Mullet and Welie *et al* refer to.

Table 2.8 Relationship types from CHI'03

Relationships	Description
is-a	Two patterns are alternative solutions for the same problem.
is-contained-by	A pattern is smaller and is used by larger one.
contains	A pattern contains a smaller one. (or uses it)

In summary, there is no agreement about the types of relationships between patterns. Exploring the relationships between patterns is an evolving process. Pattern relationship types may be used to express relationships between patterns in other domains. Schuemmer (2003b) believes that it requires a tool to express the structure of a pattern collection and make the relationships explicit. Visualisation of overlapping hierarchies could facilitate this discovery process and such visualisations should help both researchers and designers to more easily understand the abstraction levels within a set of patterns and assist in using links when identifying which patterns are required to solve a specific UI design problem. Chapter 3 will investigate methods to support the exploration and accumulation of relationships.

2.4 Pattern Forms

While many different domains have adopted the idea of the pattern, the form the individual patterns are presented in varies within these domains. A comparison of HCI pattern structures is shown in Table 2.9. In comparison with other domains, patterns related to HCI appear to have more varied structures. Furthermore, such variation can also happen within the same collections.

Although slightly different from each other, these HCI domain pattern forms are all based on the Alexandrian problem-solution pattern structure within a given context. They just extend, reduce or combine some elements of, the Alexandrian form. For example, the body of narrative description for the pattern in the specific Laakso form (Laakso, 2003) is used to explain what the pattern can do in context, thus, it has the essence of the Alexandrian pattern form. All of the pattern forms detailed in Table 2.9 emphasise that patterns are connected to other patterns. Alexander *et al.* (1977) consider this relating of patterns necessary so that a user can grasp a collection of patterns as a whole.

Only the Portland Form (Cunningham, 1994) follows the typographical rules of the Alexandrian form for presenting a pattern's content. Other HCI pattern forms use explicit labels to identify each part of a pattern. There are various ways of referring to the different

parts of a pattern. For example, the label "use when" matches the label "context", the "why" label matches the "forces" label and the section labelled "how" matches the label "solution".

Table 2.9 Comparing the major pattern forms from different domains

Name & Source	Domain	Sections within the pattern description	Types of Element
Duyne Form (Duyne *et al.*, 2003)	HCI (Web application)	name, picture, background, problem, forces, solution, and other patterns to consider	text, pictures, diagrams
Granlund Form (Granlund *et al.*, 2001)	HCI (Task or design pattern)	name, context, problem, example, forces, design solution, resulting subtask patterns, resulting structure & navigation patterns and resulting GUI design patterns	text, pictures
Laakso Form (Laakso, 2003)	HCI (User interface)	name, a body of narrative description for the pattern, examples and references	text, pictures
Portland Form (Cunningham, 1994)	HCI, software engineering	name, problem, context, forces, solution, rationale, resulting context	text, source code
Schuemmer Form (Schuemmer, 2003a)	HCI (Groupware)	name, confidence, thumbnail, also known as, problem, scenario, diagnosis, solution, participants, rationale, implementation issues, known uses, related patterns	text
Tidwell Form (Tidwell, 1999)	HCI (User interface)	name, examples, context, problem, forces, solution, diagram, resulting context, notes	text, diagram, pictures
Tidwell Form (Tidwell, 2002)	HCI (User interface)	name, picture, use when, why, how and examples	text, pictures
Welie Form (Welie, 2004b)	HCI (GUI and MoblieUI)	name, author (sometimes), problem, principle (sometimes), context, forces (sometimes), solution, rationale, examples, known uses, related patterns and implementation	text, pictures, source code
Welie Form (Welie, 2004b)	HCI (Web application)	name, picture, problem, use when, solution, why, more examples, known uses and comment	text, pictures

In 2001 Borchers claimed that there could be a high level of agreement within the HCI community on defining a common structure for HCI patterns but it was not until 2003 that the first version of a common form named PLML was agreed upon. The definition of PLML should facilitate the development of a more complete HCI pattern language (Fincher, 2003). There is a description of PLML in the next section.

2.5 Formal Pattern Forms for HCI Patterns

This section introduces the formal pattern forms for specifying HCI patterns, such as XML, Interface Markup Languages, and PLML.

2.5.1 XML

Extensible Markup Language (XML) is a universal format to document structured data and semi-structured data. One significant feature of XML is that its vocabularies are related to the meaning of the XML document. The concepts, elements and attributes within a XML document can be considered as the vocabulary of that XML document. Understanding the vocabulary in a XML document properly makes it possible to compare documents and convert between different structures. XML allows users to create their own vocabulary based on need and it provides a mechanism to formalise the vocabulary using either a DTD (Document Type Definition) or a XML Schema. DTD was the first technology that allowed users to design their own XML files based on their own tags. However, DTD is not written in XML elements and it does not support data typing. In contrast, XML Schemas are written using XML elements and attributes. Compared to DTDs, they are easier to understand. XML Schemas support data typing which allows the schema author to define actual data types or to specify a range of values (Aiken & Allen, 2004). Therefore, users can define a new language for a specific purpose such as user interface design.

2.5.2 Interface Markup Languages

XML-based device independent interface languages have been described by different research groups: Extensible Interface Markup Language (XIML.Org), Extensible User Interface Language (Mozilla), User Interface Markup Language (UIML.Org), Alternate Abstract Interface Markup Language (AAIML) (Zimmermann *et al.*, 2002), Abstract User Interface Markup Language (AUIML) (Merrick, 2001), Microsoft Extensible Application Markup Language (XAML), User Interface eXtensible Markup Language (UsiXML.Org), and W3C XForms (W3C.XForm). The details of each interface markup language can be found in http://xml.coverpages.org/userInterfaceXML.html#auiml. Four of them have been selected to explain in detail because they are discussed frequently by researchers and applied in the implementation of tools such as IdealXML (Simarro, 2005) and KnowiXML (Furtado *et al.*, 2004) both building on UsiXML.

XIML is an XML-based language for representing user interfaces and allowing universal support of functionality across the entire interface design lifecycle: design, operation, management, organisation, and evaluation (Puerta & Eisenstein, 2002). XIML organises interface elements into interface components. It predefines five basic component types: task,

domain, user, dialog, and presentation (Puerta & Eisenstein, 2002). The first three components are abstract and the last two are concrete.

XUL was developed by the Mozilla group, and it can be parsed and displayed in the Mozilla browser. This interface markup language adopts and supports many web technologies: XML 1.0 (W3C.XML), HTML4.0 (W3C.HTML), Cascading Style Sheets (W3C.CSS), Document Object Model (W3C.DOM), and JavaScript (Netscape). The combination of these technologies makes XUL very powerful.

UIML is a meta interface markup language, which requires adding vocabulary into itself. Vocabulary can be created to capture UI metaphors, author intents, describe controls, and so on. UIML tries to represent any kind of UI by normalising the different syntaxes into its own syntax. It divides a user interface into six parts: structure, style, content, behaviour, APIs to components outside the UI and mappings to UI toolkits.

UsiXML is another XML-compliant markup language, which supports the variety of interfaces like Character User Interfaces, Graphical User Interfaces, Auditory User Interfaces, and Multimodal User interfaces. UsiXML is a platform and device independent language and it describes the UI elements at a high-level of abstraction.

All these four XML-based interface languages are relatively easy to transform and use across platforms. They separate content from structure thereby make it simpler to modify and customise UI designs. As well, they separate the business logic and presentation logic so can significantly reduce the interference between user interface designers and programmers.

However, these interface markup languages cannot be used to present HCI patterns directly as they are not designed for this purpose. None of them have provided specific tags to document HCI pattern content. But, the interface markup languages can be used to present and specify instances of HCI patterns (Sinnig *et al.*, 2003).

2.5.3 Pattern Language Markup Language (PLML)

Welie *et al.* (2000) first used XML to create a standard and consistent pattern format for publishing HCI patterns in their pattern catalog website. Later, some researchers advised the use of XML to document patterns for pattern tools (Borchers, 2001; Greene *et al.*, 2002). CHI 2003 ran a workshop that is critical in this development as the members agreed on the

specification of the first version of PLML (Fincher, 2003). PLML uses XML to document pattern data content in a consistent form based on the standardised XML DTD or Schemas (Appendix A). PLML can be used to unify the format of existing pattern languages from different collections. PLML includes the following elements (Fincher, 2003):

> *<!ELEMENT pattern (name?, alias*, illustration?, problem?, context?, forces?, solution?, synopsis?, diagram?, evidence?, confidence?, literature?, implementation?, related-patterns?, pattern-link*, management?)> (p.27)*

Each of these elements (Fincher, 2003) is now discussed in turn. Name is the identifier of a pattern, which contains simple, meaningful and understandable words. A good name may reveal the essential concept of the patterns that helps people understand and apply patterns easily. In PLML, it is presented as the element <name>. Fincher recommends it should contain fewer characters. However, there are different names for a pattern in reality so "Alias" is used to contain different names for patterns in their previous descriptions using the <alias> element.

PLML uses the <illustration> element to represent a good example of the pattern in real life, while in other schema this is usually referred to as a picture (Fincher, 2003, p.26). This element provides a place to explain a direct instantiation of the pattern to help people grasp what the pattern presents from an empirical aspect.

The element <problem> is used to describe the design situation that appears in the software development process again and again. It is very important for readers to read and understand what the pattern will do for them. Therefore, readers can see if the pattern matches the design problem they require a solution for.

The <context> illustrates the applicability of the pattern and defines the proper preconditions for applying it. It formalises the situations in which the pattern can be applied properly.

Forces are used to describe all relevant constraints upon the pattern and the trade-off relationships between them. As a very important part of pattern description, they can help readers understand and consider some known trade-off effects on the goal once the pattern is applied. More details about forces will be given in the next section.

The solution contains a set of rules and instructions to assist readers realise the desired design result. The solution can include narrative description, graphics, etc.

The element <synopsis> in PLML describes the summary of the pattern, which describes the pattern briefly at a high level.

A diagram explains the detailed content of the patterns in a schematic form to readers. The diagram can be either a formal representation or just an informal sketch.

An element <evidence> and the two sub-elements (<example> and <rationale>) of <evidence> in PLML are used to present the common concept of examples and rationale in the pattern description. Examples represent current good solutions for the pattern, which include known uses. The main purpose of examples is to show the reader some prior experiences with the pattern usage and help the reader to capture the essentials of the pattern in a short time. In the meantime, it provides a way to store some excellent examples for researchers or designers to make improvements in the design process. The rationale indicates how the pattern works. It looks deeply into the structure of the pattern and explains how the solution solves the problem related to forces and context.

The element <confidence> in PLML indicates the level of the belief that applying the pattern truly resulting in a proper solution. PLML uses the element <literature> to record any related references. The user may search for related work from those references. The element <implementation> is used to store important information during the pattern design process. For instance, it can record code fragments, or other detailed implementations of the pattern.

Individual patterns play an important role in pattern languages. Therefore, a pattern needs to keep the information about its relationship with others. PLML uses <related-patterns> to define the relationship between patterns in a collection or among collections. The related patterns are presented as <pattern-link type= "" patterned= "" collectionID= "" label= "">. The three types of link: (is-a, is-contained and contains) recommended in the PLML definition where discussed in Section 2.3.3.

PLML also contains some elements for management purposes: <author>, <credits>, <creation-date>, <last-modified>, and <revision-number>.

2.6 Forces in the Pattern Form

PLML has briefly introduced forces (Section 2.5.3). Forces are an essential part of HCI pattern descriptions and can be classified by importance and type.

In the Alexandrian pattern form, the forces are subsumed within the problem statement and visualised in the solution diagram. The problems statement is used to help the pattern reader understand the trade-offs in the design problem. Alexander outlined the importance of forces in the preface to paperback edition (1971) with the quote:

> *The idea of a diagram, or pattern, is very simple. It is an abstract pattern of physical relationships which resolves a small system of interacting and conflicting forces and is independent of all other forces, and of all other possible diagrams (p.ii).*

This quote implies that, if a pattern reader wants to understand the problem and solution of a pattern, then we have to understand the forces, the conflicts and the interactions among them with respect to the design goals. The rationale of a pattern can be grasped through understanding the forces. Therefore, forces are a key element for understanding a pattern which confirms Coplien's view (1996) that forces are the focus of a pattern. Salingaros (2000) also believed that the pattern is an encapsulation of the forces that the solution to the problem has to deal with.

Henniger (2001) says that the forces "shape the problem and its variants". Understanding the forces will help designers apply patterns efficiently. O'Callagham tells us that the solutions for design problems need to resolve the same forces every time while they are applied in different contexts, in the introduction chapter of Graham's book (2003).

Duyne *et al* (2003) emphasised that the forces are the key when trying to resolve a design problem. Once the user understands the forces of a pattern, he or she will grasp the difficulty of the problem that a pattern addresses. Then the user will know how to balance the trade-offs in the solution to the problem. Sometimes, opposing forces will point out what will not work and indicate what will work in the solution (Coplien, 1996).

Alexander observes that "every design problem begins with an effort to achieve fitness between two entities: the form in question and its context. The form is the solution to the

problem; the context defines the problem" (1964, p.15). He continues: "fitness is a relationship of mutual acceptability between these two" (1964, p.19). Fitness, therefore, is one kind of force that can extend and elucidate the problem statement. Actually, there are good fits and bad fits in real life. However, it is more difficult to identify the fit than the misfit that prevents a good fit. Alexander writes: "...These misfits are the forces which must shape it, and there is no mistaking them. Because they are expressed in negative form they are specific, and tangible enough to talk about" (1974, p.23).

The Alexandrian pattern includes psychological, social, and economic forces as well as physical aspects (Alexander, 1979, p.108-110). The HCI community adopted this theory and some HCI pattern writers have explicitly classified forces using a similar system. For example, the PSA (Pattern-Supported Approach) to the user interface design process divides forces into task, user and context forces (Granlund, 2001). The task-forces represent the characteristics and special requirements for the task and the user-forces are used to illustrate a picture of a typical user. Further, the context-forces capture environmental and social factors that will affect the design. Duyne *et al.* support this point even though they do not explicitly classify forces in their patterns: "the forces follow the problem, describing it in more detail, examining how people, their tasks, the technology, and society affect the design problem" (2003, p23).

In summary, forces have been emphasised by different researchers in the domains of architecture, software engineering and HCI. In two main collections from the HCI domain, the discussion of forces is often the longest section of each pattern (Duyne *et al.*, 2003; Graham, 2003a). The forces are the key to understand the essence of a pattern, and can help UI designers browse for the most appropriate pattern solutions for interface designs. Researchers can also use the forces to identify relationships between patterns and to further categorise patterns.

2.7 Categorisation of Patterns

While many individual patterns and pattern collections have been created, it requires some principles to organise the patterns into collections or pattern languages, which help the pattern reader comprehend the patterns and the relationships among them. Classification can assist the designer in browsing and selecting patterns effectively. Categorisation makes it more efficient for choosing and combining patterns to solve a complex design problem.

Different organising principles have been discussed by some researchers. Various principles deal with different purposes. Some principles discuss the way to organise patterns into a pattern language (Fincher *et al.*, 2000; Mullet, 2002; Welie & Veer, 2003). Others are proposed to group patterns into collections or meta-collections (Mahemoff & Johnston, 1998a; Tidwell, 1999; Welie & Trætteberg, 2000; Tidwell, 2002). Below, there is a brief introduction to some of the different classification schemes.

Michael J. Mahemoff and Lorraine J. Johnston (MJ)

Mahemoff and Johnston (1998a) investigate how patterns can be used in the development cycle. Their classification scheme identifies four kinds of patterns in the HCI domain. These are:

- *Tasks. This category of patterns captures the tasks that users will perform in interaction with the system.*
- *Entire Systems. Patterns of entire systems describe overall issues, which explain the high level of abstraction of the system.*
- *User-interface Elements. The patterns of this type express the usage of user interface elements.*
- *Users. Patterns capture the profile of users.*

ChiliPLoP'99 (C99)

Another categorisation was introduced at ChiliPLoP'99 (Borchers, 1999). This categorisation is based on three dimensions: the level of abstraction, function, and physical dimension. The main dimension is the *level of abstraction*, which is divided into three parts: task, style, and object.

The second dimension is *function*, which deals with the classification of patterns related to functions. This dimension includes the following functions: perception, manipulation, and navigation.

The third dimension in the taxonomy is the *physical dimension*, which explains how the patterns express the temporal and spatial questions. This dimension contains three attributes: space, sequence, and time.

Jenifer Tidwell (JT)

Tidwell's patterns (1999) are divided into four primary categories based on two guiding goals. The first goal is to shape and represent content to the user in a proper way and the second is to make actions available for the user. The categories are:

- Basic shape of content
- Basic shape of actions
- Unfolding an artefact's content and actions
- Others -- these are sets of patterns that deal with: use of space and other resources, navigation, specific actions and interrelationships.

All these categories include sub-categories, which normally refer to lower level patterns.

Tidwell actually classifies her patterns using an alternative categorisation which divides the patterns into ten categories based on a set of questions that a designer might ask when creating a new interface:

- What is the basic shape of the content?
- What is the basic shape of the actions taken with the artefact?
- How do the content or available actions unfold before the user?
- How does the artefact generally use space and the user's attention?
- How is the content or action organised into working surfaces?
- How can the user navigate through the artefact?
- What specific actions should the user take?
- How can the user modify the artefact?
- How can the artefact be made visually clear and attractive?
- How else can the artefact actively support the user?

In Tidwell's new pattern collection (Tidwell, 2002), patterns are grouped into eight sub domains:

- Organising the Content
- Getting Around
- Organising the Page
- Getting Input From Users
- Showing Complex Data
- Commands and Actions

- Direct Manipulation
- Stylistic Elements

Sally Fincher (SF)

Fincher (2000) described the organising principles for constructing pattern languages that can help identify and select patterns for practical use at the CHI 2000 patterns workshop (Fincher & Windsor, 2000). The first principle is called the *scale* principle, which is derived from the Alexandrian structure of scale and the second principle is based on the *tasks*. The other two categories are based on information and *context* and *society*. This taxonomy can also be used to categorise the different phases of the development process such as the: analysis space, problem space and solution space.

Martijn van Welie and Hallvard Trætteberg (WT)

Van Welie and Trætteberg (2000) claim that the categorisation of pattern collections will aid researchers and designers in their understanding of both individual patterns and whole collections of patterns. As a result, understanding patterns precisely should ensure the proper application of the patterns in a practical design. Therefore, they want to construct an optimal structure for pattern collections to help designers search for target patterns to address a design problem. Van Welie and Trætteberg's patterns are task related patterns. Thus, they propose to classify those patterns based on usage problems. The underlying user interface principles originated from Norman's *The Psychology of Everyday Things* (1988). Each principle is a classification category in their scheme. Norman's principles are summarised below (Welie & Trætteberg, 2000):

- *Visibility*. Gives the user the ability to figure out how to use something just by looking at it.
- *Affordance*. Involves the perceived and actual properties of an object that suggest how the object is to be used.
- *Natural mapping*. Creates a clear relationship between what the user wants to do and the mechanism for doing it.
- *Constraints*. Reduces the number of ways to perform a task and the amount of knowledge necessary to perform a task, making it easier to figure out.

- *Conceptual models.* A good conceptual model is one in which the user's understanding of how something works corresponds to the way it actually works. This way the user can confidently predict the effects of his actions.
- *Feedback.* Indicates to the user that a task is being done and that the task is being done correctly.
- *Safety.* The user needs to be protected against unintended actions or mistakes.
- *Flexibility.* Users may change their mind and each user may do things differently.

Kevin Mullet (KM)

Mullet (2002) has tried to find a way to organise and extend pattern languages that can deal with new design problems. Mullet argues that it is very easy to identify the highest level and lowest level patterns within pattern collections. But, as he points out the levels between the application-level (high-level) and the widget-level (low-level) are not easily discerned. Therefore, Mullet recommends that a framework should be constructed that is externally visible so that the logical structure linking patterns can be used by to guide in selecting patterns to solve a specific problem. He proposes a two classification schemes. The first uses the relationships between patterns to organise the patterns into interlocking hierarchies. The second scheme is to organise the patterns based on the level of design activity they might be used at.

Three types of relationship (see also Table 2.6) that Mullet (2002) proposes for linking patterns into hierarchies are:

- Derivation discusses the parent-child relationship between patterns.
- Aggregation describes the relationship between the whole and the parts.
- Association explains the cross-reference relationship between patterns.

The hierarchies created by the first two relationships are orthogonal to each other while the structure created by the third cuts across these hierarchies creating a network of patterns.

Mullet (2002) identifies three levels of design activity:

- Conceptual Design represents the organisation and content of the artefacts.
- Presentation Design describes how to make conceptual design become concrete design.
- Interaction Design discusses how the user can control the application and administrate the objects within the application.

Martijn van Welie and Cerrit C. van der Veer (WV)

Van Welie and van der Veer (2003) explore the issues related to how to structure collections of patterns into pattern languages. They believe that the strength of organising patterns into a pattern language is that such organisation should add significant value to the collection of patterns. Furthermore, one of their goals for classifying patterns is so they are available for application in practice. The different methods they identify for organising patterns are:

- Connecting all patterns into a *pattern language*
- Grouping patterns by *function* or *problem similarity*
- Grouping patterns based on *usability defect*
- Clustering patterns by *user task* and *user type*
- Classifying patterns based on *site posture*

Importantly they describe the concept of "scale" when organising patterns in the HCI domain as being about the "scale of 'problems' rather than geometry" (Welie & Veer, 2003, section 3). Using this meaning for scale they propose four levels of classification: *posture type* patterns, *experience* patterns, *task* patterns and *action* patterns. Posture type patterns describe the goal of the type of the interface, for example, the goal of a website, which is similar to "Site Genres" (Duyne et al., 2003). Experience patterns discuss the common goal and tasks expected by the users. Task patterns describe concrete ways to achieve the task goal. Action patterns describe the lowest level of tasks.

As Van Welie and van der Veer point out many pattern collections (Tidwell, 2002; Duyne *et al.*, 2003; Laakso, 2003; Welie, 2004b) have been categorised by function or problem similarity. In this form of categorisation, patterns are grouped together based on aspects of the problem. Designers can easily browse and access specific patterns for their current design problem.

The usability performance enables designers to filter patterns based on their usability defect. Categorisation of patterns can also be based on specific purposes such as focusing on patterns for a particular use or in a specific domain.

Ian Graham (IG)

For presenting and navigating the pattern language easily, Graham organises his web usability patterns into a pattern language based on the temporal order of the design process, the author organises those patterns into four categories: getting started on your site, enhancing usability, adding detail, and workflow and security. Each section of the category might include different pattern types: abstract patterns, concrete patterns, and concrete and the terminal patterns (Graham, 2003a).

Van Duyne *et al* (VD)

The book, *The Design of Sites* (Duyne *et al.*, 2003), includes a large collection of web site design patterns. The authors classify those patterns into groups, from high level site genres to low level page elements. This assists the users in browsing the patterns efficiently. The users can navigate the pattern collections quickly and select the target patterns for solving their design problems.

Mary Zajicek (MZ)

Zajicek (2004) collected a set of patterns for speech interaction among older adults. This collection is based on a set of forces. Although Zajicek did not mention a way to classify patterns, she does imply that it is possible to create a collection of patterns based on a common set of forces. This indicates that it is possible to group patterns based on forces.

Summary

Although different researchers have discussed and proposed different principles for organising patterns, the essential goal of categorisation is to organise patterns in pattern collections, which will help the HCI community to understand them and put them to practical use. However, some early attempts were not illustrated with examples, probably because there were not enough HCI patterns at the time (Fincher & Windsor, 2000; Mullet, 2002). A few categorisations are linked to the development process (Mahemoff & Johnston, 1998a; Fincher & Windsor, 2000). For instance, the first category of Graham's method is "getting started on your site", which includes those patterns for planning new websites. Fincher's method focuses on different phases of the design process. Mullet's method aims to

identify the different levels between patterns while van Duyne *et al* classify their website patterns by using a similar idea. Tidwell organises the patterns using questions because these questions can aid the pattern reader in matching their needs with target patterns. Categorising patterns based on forces could also identify similar or related patterns. Although some of the above categorisations cross over one another, there is no common methodology to classify and categorise patterns. It is suggested that there is a need to combine different kinds of categorisation (Welie & Veer, 2003). Some principles might be a subset of others.

2.8 Summary

This chapter has reviewed the background to patterns. The advantage and disadvantage of UI patterns and UI design guidelines have been discussed. Compared with guidelines, UI patterns appear to be more useful in guiding UI design. Many UI pattern collections have been created, but they are written in different formats. As a result, it is difficult for researchers or users to discuss the patterns using a common language. A standard pattern structure, PLML, was specified to aid this purpose. Four key issues related to patterns, have been outlined, a formal pattern form, the identification of relationships, the importance of forces, and the categorisation of patterns. Different types of relationships between patterns have been described and summarised. The analysis of categorisation has been illustrated, but there are few commonalities between different methodologies. A pattern tool is required to support these key issues.

The next chapter will review existing specifications and pattern tools, and propose a framework to support the key issues.

Chapter 3 Existing UI Pattern Management Specifications and Tools

3.1 Introduction

Computer systems have been proposed to facilitate researchers exploring the key conceptual issues relating to UI patterns that were discussed in Chapter 2, and to guide UI designers in using UI patterns. This raises the issue of what kind of support such a computer system should have. Proposed specifications for pattern tools, as well as existing tools, will be investigated to determine user needs. The following discussion focuses on the essential features a pattern management tool should have in both the creation and use of pattern collections.

3.2 Existing UI Pattern Management Specifications and Tools

As numerous pattern collections become easily available, one challenge for the HCI community is to build tools to support pattern creation and the use of patterns in user interface design (Gaffar *et al.*, 2003). Thus, many questions related to building a pattern tool need to be answered. The most important issue is what the pattern tool can do for the HCI community. Four specification definitions for a pattern tool from different HCI research groups in recent years have been identified and are described below.

Borchers' Specification (BS)

The user may wish to author and create pattern collections by using a computer system. The existing collections may be reviewed by the user. The system may be used to search and apply patterns to the current design problems by the user. Therefore, Borchers (2001) claims that a pattern editing tool should have the following functionalities:

- Creating, annotating, and reading pattern content.
- Including multimedia examples such as audio, video data and image data.
- Linking to other patterns and pattern languages.

- Creating graphical representation of the existing pattern hierarchy.
- Providing search functions including a keyword search or full text search.
- Allowing navigating around the language and printing patterns.
- Supporting various operating systems without the use of additional specific software applications.
- Adopting a widely accepted structure for patterns.

Greene's Specification (GS)

In CHI'02 Patterns in Practice workshop, Greene *et al.* (2002) presented a requirement specification for a pattern tool. Issues covered include:

- *Composition* defines the creation of patterns based on a flexible template with facilities to get content from other resources.
- *Linking* provides support for different types of association between patterns as well as to related pattern collections. A second form is for linking examples and supporting evidence to pattern content. This is required for connecting to the cited references, applications, software components and various media types.
- *Modification* for editing and annotating existing patterns.
- *Evaluation* to support the pattern rating. There should also be a mechanism for keeping track of how the tool is used.
- *Browsing* provides different views of patterns and pattern collections including summaries, full details and visualisations.
- *Searching* facilities, such as keyword, full text and advanced search. Advanced search provides target pattern searching based on a set of elements within the pattern form, such as problem, context, forces, etc.

Greene *et al.* also recommend that patterns are stored using a common structure within the tool and that output should be in the form of a self-defining data structure based on XML.

Schuemmer's Specification (SS)

There are some additional requirements for a pattern authoring and reading environment described by Schuemmer (2003b): modelling patterns, artefacts, and relationship in a visual patterns map; providing interactive mechanisms to filter interested relationships; using automatic layout algorithms to display the pattern map; linking the textual and the graphical

representation of a pattern; automatically synchronising graphical and textual pattern representations.

Gaffar's Framework (GF)

Gaffar *et al.* (2003) create a framework for disseminating and sharing HCI patterns based on their seven C's methodology. The methodology aims to centralise and organise the pattern collections into a pattern repository and disseminate the repository of pattern knowledge to the HCI community:

- *Collect*: Offering a central repository to gather isolated patterns or pattern collections.
- *Cleanup*: Providing a pattern presentation format to unify differing pattern forms into the same style.
- *Certify*: Defining specific domains and collecting together patterns focusing on a domain in a collection.
- *Contribute*: Providing an open repository accessible for all in the UI pattern community.
- *Categorise*: Defining clear categories within a collection and potentially model those categories as a hierarchy of categories to reduce the complexity of discovering relationships between patterns.
- *Connect*: Defining and creating relationships between patterns within a relationship model.
- *Control*: Defining a machine-readable format so future tools can automate the UI design process using patterns.

Summary of Specifications

The above specifications focus on the activities related to pattern management. They all recommend the adoption of a common pattern form. Borchers' specification emphasises the editing of individual patterns and the representation of links between patterns as a graphical map. The tool proposed by Schuemmer focuses on the visualisation of relationships between patterns using a graph. Greene's specification gives richer details regarding the building of a pattern tool than the others. For example, this specification identifies the features of the support of alternative pattern forms, input pattern template, advanced search, creation of relationship type, pattern rating and modification of pattern. Gaffar's specification does not define any facility for visualisations of the relationships between patterns but it does require

that a pattern be a member of a specified collection and that patterns within a collection can be further categorised. Only Greene and his co-researchers propose to support the creation of new alternative relationship types. Borchers, Schuemmer and Greene all propose the use of XML, but not PLML.

None of the above specifications define pattern versioning and customisation of existing patterns. Neither do they explore the power of forces to enrich the functionality of a pattern tool. Moreover, all these specifications are defined to support the management of patterns or relationships between patterns, while there is less emphasis on the use of a tool by an interface designer.

The features identified by these specifications are summarised in Table 3.1 (Y means the specification supports the specific feature):

Table 3.1 Summary of features of four specifications

Features / Specifications	BS	GS	SS	GF
Support specific pattern form	Y	Y	Y	Y
Support alternative pattern form		Y		
Input pattern template		Y		
Create new pattern	Y	Y	Y	Y
Support multimedia data	Y	Y		
Search pattern by keyword	Y	Y		Y
Search pattern by full text	Y	Y		
Advanced search		Y		
Browse pattern		Y	Y	Y
Create relationship type		Y		
Create relationships between patterns	Y	Y	Y	Y
Visualise relationships between patterns	Y	Y	Y	
Pattern rating		Y		
Write comments	Y	Y		
Different views	Y	Y	Y	Y
Modify pattern		Y		
Categorise pattern				Y

3.3 Existing HCI Pattern Tools

This section describes a selection of tools, which support the creation, management, and use of patterns or pattern collections. Tools with similar features have been grouped together. Some pattern tools are publicly available. Users can access and use most of the web-based pattern tools. A few executable pattern tools can be downloaded from a developer's web

page, such as CoPE (Schuemmer, 2003b) and IdealXML (Simarro, 2005). No working versions of the current pattern management tools seem to fully support PLML, but some tools will probably be updated to support it. A detailed description for each selected tool is presented, identifying the functions each provides.

3.3.1 Types of Pattern Tools

Different types of pattern tool have been developed to support the HCI community in creating, managing and using patterns. Fifteen existing tools have been investigated and organised into three groups. The tools have been categorised using a "best fit" strategy. Table 3.2 categorises these UI pattern tools into three types of tools.

A *pattern catalog tool* generally has the following features: collection and presentation of patterns organised by different categories; access to patterns via the Internet; and some aid for the creation and submission of new patterns. The most basic of these tools are just web pages containing patterns categorised into lists without any pattern submission function. These tools focus on disseminating and sharing patterns through the Internet.

A *pattern management tool* provides facilities to manipulate patterns and pattern collections. First, they provide templates for pattern creation, which could be used as a common transfer form between tools. Second, these tools support searching or browsing collections of patterns and displaying patterns from different views. Third, they provide facilities for modifying existing patterns. Finally, they have mechanisms for organising patterns and creating relationships between them. These tools emphasise the creation of patterns and managing them within pattern collections.

A *pattern-based design tool* provides support for applying patterns to help HCI design activities. Some include basic pattern creation functions as well as functions for customising patterns for an application domain but a number just use a pattern repository without providing any pattern management functions. These tools have automated some processes in user interface design, such as, creating storyboards or generating code. These tools emphasise the use of patterns for designing user interfaces for specific applications.

We have investigated a selection of pattern tools from the three different categories when researching the requirements for a tool for managing collections of UI patterns. The features

supported by these tools are identified. Some functions are already supported by some tools, such as the creation of the relationships between patterns. However, some important functions get less support, such as managing collections, pattern versioning and alternative categorisations and manipulating forces. These tools are described in the next section.

Table 3.2 Summary of tools *("w" indicates proposed functionality)*

Category / Developers	Catalog only	Management	Design
Welie (2004a)	PID		
Tidwell (1999)	CG		
Tidwell (2002)	UIT		
Laasko (2003)	UIDP		
Martin *et al* (2004)		PoInter	
Graham (2003b)		Wu	
DIAC (2002)		PSP	
Gaffar *et al.* (2003)		MOUDIL	
Borchers (2001)		PET	
Schuemmer (2003b)		CoPE	(*w*)
Greene *et. al.* (2003)		IBM	(*w*)
Henninger (2001)			GUIDE
Lin & Landay (2003)			Damask
Henninger *et al.* (2003)			SWT
Molina & Hernandez (2003)			ONM
Simarro (2005)			IdealXML

3.3.2 Cataloging Tools

Pattern in Interaction Design (PID)

The goal of PID is to create and publish a collection of UI patterns that deal with the problems of end-users. The tool aims to allow any user to submit new patterns and the patterns will be discussed by a small group of researchers and other users. A standard UI pattern format is proposed to construct the tool using XML. PID is the first tool to adopt XML to document patterns (Welie *et al.*, 2000).

This pattern cataloging tool (Welie, 2004a) is basically a website. It publishes patterns in groups based on three application domains: Web Design patterns, GUI Design patterns, and MobileUI Design patterns. Within each group, patterns are further categorised into low-level groups. The Web Design patterns group contains 113 patterns, which are divided into 10

sub-categories. The patterns are categorised in groups by function or problem similarity, a method proposed by van Welie and van der Veer (2003) that was mentioned in Chapter 2.

There are two different pattern formats used in the tool, although the developers claimed to have adopted a unified pattern format for representing their UI patterns. The Web Design patterns include the following components: pattern name, illustration, problem, use when, solution, why, more examples (including description and image), known uses and comments. The other two groups of patterns are represented in a similar pattern form: pattern name, author, problem, principle, context, forces, solution, rationale, examples, known uses, related patterns and implementation. However, the MobileUI Design patterns may not contain information such as principle, forces and implementation, as it has a briefer pattern form compared to the GUI Design patterns.

The user can read the detail of each pattern and add their comments to the bottom of most patterns. There are links to related patterns, cited literature and other resources supporting a pattern, using standard hyperlinks. Currently, there is no pattern submission facility for visitors to the site.

Common Ground (CG)

Common Ground (Tidwell, 1999) is composed of a number of web pages, which organise and categorise a collection of patterns using questions that address specific design problems, to assist users in choosing the patterns they wish to view. The underlying methodology for categorising these patterns has been described in detail in Chapter 2. Tidwell considers the collection of patterns as an Alexandrian pattern language. The goal of the tool is to create and publish the pattern collection in this tool to make it available to the public. It will enable UI designers to produce good quality interfaces because designers can apply the pattern collections to their design immediately. Designers can access the tool through the Internet to find an appropriate pattern solution for solving current design problems based on the context. The tool also provides a common UI language for designers to communicate with different stakeholders.

All patterns in this tool are presented by using the same structure: pattern name, examples, context, problem, forces, solution, resulting context and notes. This pattern structure is similar to the Alexandrian pattern form but it separates the problem statement and forces

while they are described together in Alexandrian patterns. Any review opinions and comments regarding these patterns can be sent to the author directly. Some additional information has been appended to some patterns as notes. This tool relies on Internet browser facilities to search for specific patterns and standard hyperlinks are used to navigate to related patterns.

UI Patterns and Techniques

This tool (Tidwell, 2002) presents a new collection of UI patterns, which continues the work of the patterns illustrated in the Common Ground tool. Compared to the patterns of the Common Ground, the patterns from this collection are described in a brief and different pattern structure, including only the sections: pattern name, illustration, use when, why, how and examples. The pattern form shows the essential idea of a pattern in a simple and direct way that can facilitate understanding for UI designers or even end-users. Another major difference is in the way the collection of patterns has been grouped. The patterns are categorised and ordered into eight different groups depending on common design problems such as "Organising the Content" and "Getting Around". The author of the patterns implies that the tendency for desktop application and web application interface is for them to become similar in near future. Therefore, the patterns within the collection are not grouped based on the types of application. The author believes that UI designers may adopt wisdom from various types of application to improve current interface design. Internet tools provide the browsing and searching facilities for the repository.

User Interface Design Patterns (UIDP)

UIDP is similar to the previous three in that it is basically a website. The distinguishing feature is that the set of UI design patterns reflect recurring design problems based on the users' goals (Laakso, 2003). The patterns try to outline the recurring design problems encountered when designers make an effort to produce a good UI design. The author of the patterns emphasises that the method used to produce good interfaces is based on a simulation of the user's goals.

The Alexandrian pattern form is not used in the description of the patterns within this tool. The specific pattern structure includes four components: pattern name, a description of the

pattern, examples and references. The author of the patterns tries to use the pattern structure to outline the most important findings for each pattern.

3.3.3 Management Tools

PoInter

PoInter (Martin *et al.*, 2004) currently presents a small collection of 10 cooperative interaction patterns. The tool provides a resource of patterns for UI designers to use while they are analysing and prototyping cooperative systems.

The website uses standard WIKI technology to provide facilities so that users can participate in the development of the pattern collection. It provides two specific pattern templates so users can create new patterns. One template is used to present patterns themselves while another template, named vignette, can be used to describe more particular details regarding how the patterns are instantiated in real situations. The vignette patterns can be accessed through the link part ("where used") of the original patterns. PoInter also allows users to modify existing patterns and annotate patterns with comments. All changes are logged. It provides simple browsing functions based on either a title index or a word index. An alternative way to facilitate the browsing function enables users to label each pattern page into different categories so that users can browse all pattern lists by clicking the name of categories. This method can be considered an alternative way to classify patterns by labelling. A basic search facility is available to find patterns depending on the title search or the any text search. One of the problems with this site is that a lot of extraneous pages have been added making browsing somewhat confusing.

Wu Website (Wu)

Wu (Graham, 2003b) is also a web-based tool which uses basic web page development technologies to provide enough functionality so that this site has been classified as a web-based pattern management tool. This tool mainly introduces a pattern language (collection) for producing web sites of good quality. The pattern language was originally published in the book, *A Pattern Language for Web Usability*, written by Ian Graham. The Wu website aims to create a research community that will evolve the pattern language and put it into practice to achieve good web usability (Graham, 2003b).

The standard browser and searching facilities have been extended so that the user can either navigate around the pattern collection using one of four visualisations of the language map, or via an alphabetical list or by a pattern's numeric identifier. Eighty patterns are grouped into four visual language maps where each pattern presents as a node with a name label and the arrow lines indicate the relationships between patterns. The patterns are further differentiated by colour into three types: abstract patterns, concrete patterns, and concrete and terminal patterns. All patterns from the repository can be presented in an alphabetical list or ordered in a numeric list for accessing. The user is allowed to read the detail of a pattern by clicking the node of the map or the pattern name. Related patterns are linked together using the web hyperlinks.

This tool adopts a pattern form close to the Alexandrian pattern structure. A simple online facility provides a template, depending on the pattern form, with some guidance so that users can create new patterns and submit them for inclusion in the language. The author of the language will review the submitted patterns before they are published on the web sites. It allows the user to edit the main content of the submitted patterns in Rich Text Format by using a Rich Text Editor. The user is encouraged to submit comments and criticism and to submit questions via email regarding the language. The site also offers advice on how to use the pattern set to design a solution to a specific UI design problem.

Public Sphere Project (PSP)

The Public Sphere Project (DIAC02, 2002) is an online repository where public users can browse and search for patterns. The project tries to identify the patterns of information and communication that can help people apply creative wisdom to build good information and communication systems. It provides a proprietary search function that finds patterns based on the content of selected pattern element fields. A comprehensive export function is used to help all users export selected patterns with selected element fields into an XML file. It allows registered pattern writers to submit and store patterns using the defined format. All submitted patterns are organised and integrated into a coherent pattern collection. There is also an administrator category providing site management facilities. The tool can retrieve all patterns from the repository and display their names in an access list, which currently contains 450 patterns. Although this repository contains some patterns that can be categorised as UI patterns, most of the existing patterns deal with social issues.

MOUDIL

MOUDIL (Gaffar *et al.*, 2004) is an online system which has been classified as an integrated pattern management environment, but much of the site is still under construction. MOUDIL was created with two aims: first to provide UI developers with access to a comprehensive set of UI patterns; secondly, to act as a forum where researchers can explore pattern collections in order to discover more about the nature of such collections. If completed, this tool should support all functionalities of the seven C's methodology as discussed in Section 3.2.

Five browsing modes are present: alphabetical order, application type, contributor, most recent, and top 20. However, only alphabetical order browsing is functioning. Currently the tool contains a small number of patterns, which are displayed as a list, ordered by name. The pattern format contains a field for linking to alternative media types, but neither example worked. There is also a field for an implementation code, which should be downloadable, but there were no examples with which to test this feature. The search facilities have been implemented and include two kinds of search: keyword search and advanced search. The keyword search can only search patterns based on the three elements of pattern structure: pattern name, context of use and usability problem. The user is required to choose one of these options and enter the search data to conduct the search. The advanced search can search target patterns based on any element. Some search element fields are attached with filtering options. The input search data for both search functions can be keywords or free form text.

Three categories of user are defined: the visitor; the registered user and pattern authors. General visitors can visit the web site and search the patterns from the repository and browse the pattern catalog in detail. Registered users can collect a set of existing patterns from the list into their own online personal space where personal notes regarding the patterns can be written and recorded. The contributors have some extra privileges. They can see some additional information and submit new patterns based on the template. The submitted patterns from contributors will be reviewed and published on the site, if accepted.

There is also a discussion forum where users can share their ideas on patterns. All kinds of users can join in the discussion by posting a message into the forum as well as viewing messages from other users.

PET

PET was proposed at the end of Borcher's book (2001). The author of the book aims to build an authoring and browsing system to write, read and browse a pattern language based on a formal model (described in Chapter 2), which is similar to the visualisation of the language map on the Wu Website (Borchers, 2001):

- The pattern language is presented in a visual map with patterns as nodes, and the relationships between patterns as directed edges or links.
- Each pattern node consists of a sequence of content blocks, which contain all the pattern contents, including text, graphics and other media data.
- The context and references blocks contain bidirectional links to other patterns within their textual contents.

The system is supposed to be used by the variety of users: HCI designers, software engineers, end-user representatives and other users. According to the description from the book, to create a new pattern the author prepares the content in XML format with external tools before submitting it. The system parses the submitted patterns and recognises different pattern contents for displaying clearly. When displaying a pattern a user can choose to show or hide any detailed content. The system will generate and present a graphical map representation of a collection of linked patterns. The content of a pattern can then be displayed by selecting the target pattern from this visualisation. PET provides facilities for reviewing existing patterns and adding personal annotations.

CoPE

CoPE (Schuemmer, 2003a) is a groupware application and a user can use it to cooperate with others online. The system also allows the user to work in individual mode, without connecting to the remote central server. This pattern management system visualises pattern collections as a semantic pattern map. The major difference between this semantic pattern map and the previous two pattern language maps (mentioned in Wu and PET) is that additional labels for relationship names are appended into the links between patterns. The essential idea in visualisations of pattern maps originates from the Alexandrian pattern language framework. The main goal of the semantic map is to help readers gain a good overall understanding of a pattern language and the map can serve as a navigation tool to link to details of pattern content by accessing the notes on the map. CoPE has been

developed to support pattern writers in two ways: first, it enables the pattern writers to create and present the patterns in a semantic pattern map; secondly, it aims to keep a track of unfinished parts of patterns.

CoPE is composed of two parts: a form-based interface pattern editor and a pattern navigator. The pattern editor is used to edit the pattern itself and provides assistance to pattern writers with linking patterns and formatting output. All pattern names are presented in an alphabetical list, differentiating the types by graphical labels. The current version of the system divides the patterns into two major groups: external patterns or patterns created in the system. The graphical labels "E" and "P" are used to mark the two different types of patterns. There are two views within the pattern editor for displaying details of a selected pattern: HTML view and editor view. The HTML view is used to show the full content of a pattern and provides hyperlinks to navigate to related patterns. The editor view separates the main element fields into different editable tabs and pattern writers can enter the content of a pattern based on the template. The tool may support different types of data as pattern content, but it cannot put illustrations into the appropriate field, as yet. The system provides a specific WIKI syntax to create the links (relationships) between existing patterns. For example, the user can build a link from a current pattern "Main Interface" to another pattern named as "Login" with the "Uses" relationship. The sample for the link syntax is: **p:Login:Uses**. Then the hyperlink between them is generated automatically in the HTML view. If both of the patterns are placed in the working area of the pattern navigator, the system will automatically create and visualise the link for them. There is a facility for rating patterns, which contains a rating scale ranging from zero stars to four stars. The user can move the Trackbar to choose a rating result for the pattern writers' confidence, which is shown in the place next to the name of the pattern, both in the pattern editor and navigator.

The pattern navigator can generate a graphical representation of the relationships between patterns. Patterns are presented as movable boxes in the navigator and relationships as labelled directed edges. Multiple relationships can be created between any two patterns. The system provides options to display the direction of the relationship edge or not. The user is allowed to name and edit any relationship type for the relationships even without naming a relationship type. There is no default relationship type available for the user to choose so the user has to decide the relationship type based on his or her own knowledge. CoPE provides a layout mechanism to organise the patterns in the graphical map separately and the user can also further adjust the map manually. If the user moves the current cursor over a pattern, a

description including "Intent", "Problem" and "Solution" of the pattern is displayed so that the user can have a quick view regarding what the pattern is about. The detailed content of a pattern from the map can be shown in the pattern editor by clicking the pattern name from the map. Furthermore, the system allows the user to create different graphical maps based on need and these graphical diagrams can be exported in Postscript or SVG format.

The developer of CoPE has a proposal for importing pattern collections based on the XML format. The user is also allowed to export the pattern collection either in XML or LaTex documents for various purposes (Schuemmer, 2003a). However, none of them work properly in the currently released version of the tool. The developer of the system further claimed to provide the functionality for manipulating the content of patterns according to the PLML specification.

IBM Tool

The IBM tool (Greene *et al.*, 2003) was developed with two major goals. Firstly, to provide a method to help pattern researchers convert and express any UI design problem in a way that allows them to explore and evaluate the "applicability" of a set of selected patterns for the design problem. Secondly, to help the UI designers or developers browse pattern sets to support both the design and development of user interfaces.

The system supports many general features as discussed in other tools such as pattern creation, and browsing, reading and editing patterns. The distinguishing feature of this tool is the addition of a "decision-support help filters". When an author creates a new pattern the author has to supply a set of criteria or drivers that define when a pattern should be used in a design. These filters can then be used by the system to guide a developer when selecting patterns to solve a specific UI design problem. The resulting decision tree can also be visualised graphically.

The description of the prototype identifies four features: the pattern editor, the search facility, a search advisor and a properties selector. Although a template is provided for creating new patterns, this is extensible and the authors suggest that this feature will result in the pattern format evolving over time. Patterns are stored and documented as XML files within the system.

3.3.4 UI Design Assistance Tools

GUIDE

GUIDE (Henninger, 2001) is a tool to help developers create more usable interfaces by applying previous experience and known best practices. It has a searchable repository of usability guidelines, patterns and cases, which are examples or "context-specific instances" of patterns or guidelines. A rule-base uses questions to lead the developers through a set of design decisions based on the project's characteristics. This process builds a navigation hierarchy to visualise the project's characteristics linked to recommended patterns and guidelines that may provide solutions. This is an interactive process and designers can submit recommendations to the system librarian to update the knowledge base, thereby improving the design advice given to the next project. This tool only uses patterns rather than providing facilities for managing pattern collections.

Damask

Damask (Lin & Landay, 2003) is a pattern-based design tool for early-stage design and prototyping of multi-device user interfaces. The proposed interface of the system includes three main regions (see Figure 3.1). The left part is a canvas where the user can sketch the interface design. The patterns from the repository can be used in the design, linking by the pattern name. The canvas can contain multiple tabs for viewing different interface designs. The user can split the canvas into multiple windows as well. A Pattern Explorer is used to browse a pattern from the repository of the system for actual interface design and the detail of a selected pattern displays in the Pattern Sidebar. UI designers can use the system to create storyboard prototypes. Damask contains a catalog of patterns from which a designer selects patterns as they build their design. The designer then customises the patterns to match the problem domain. Each design pattern includes many different examples illustrating how to apply the pattern. As well, each pattern provides a general solution for each device (PC, cell phone, PDA) supported. When a storyboard sketch for one interface has been completed the system can create an equivalent UI design for other devices. Damask automatically updates the example section of a pattern when new designs are added to the repository, but the solution must be illustrated using DENIM's sketching language.

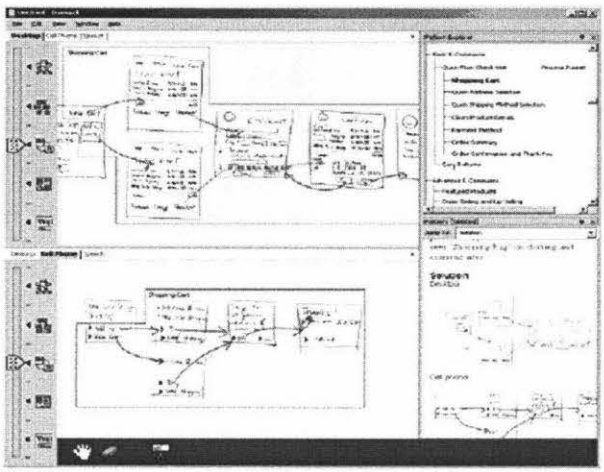


Figure 3.1 Damask's proposed user interface (Lin & Landay, 2003).

The Semantic Web Tool (SWT)

This is one of a series of tool descriptions published by Henninger (2001; Henninger *et al.*, 2003). The main goal of the prototype is to build a framework that enables users to capture and consume the patterns based on Semantic Web technologies. The architecture of this tool has three parts:

- A pattern ontology representing patterns using Semantic Web technologies (Prototégé).
- A rule based design agent which can use the pattern ontology for reasoning. This agent helps the design define the parameters for a new UI and is similar to that described for GUIDE (Henninger, 2001).
- A pattern validation component, which is a process similar to that described by Green *et al.* (2003).

OlivaNova Modeler (ONM)

ONM (Molina & Hernandez, 2003) is a CASE tool which includes a pattern-based UI design component. The system conceptualises the model of a business application and generates a programming code based on the "conceptual user interface pattern language". A model of the UI can be developed based on their UI pattern language, which includes patterns at different levels in the design process. The conceptual UI patterns have to be written using their pattern composition language, which enables the automatic validation of potential designs. A transformation engine is used to translate the specification into an implementation

code in the selected programming language and for the specified environment, such as Web, Desktop or PDA.

IdealXML

IdealXML (Montero *et al.*, 2005) is a pattern oriented CASE tool, which allows the user to edit different notations from the software engineering and human computer interaction disciplines. The system aims to manage a repository of UI patterns and provides additional functions to enable the designers to apply these patterns to an actual interface design.

The system adopts a PLML data structure as the base to document the pattern content while there are some additional elements as options. However, some key elements from PLML have not been included in the system, such as the "implementation", "pattern-link" and some sub-elements of "management", so it seems to support PLML only partially. The system divides and manages forces into four dimensions: strengths, weakness, opportunities and threats. IdealXML manages pattern data in file documents, which are manipulated by operating the system's file system. All documents of a pattern stored inside the folder are named with pattern names within the default folder "Repository". There are three main types of documents used by the system. The descriptive content of a pattern is saved as a XML file in the PLML data structure. The diagrams of notations are stored as files in UsiXML format. A variety of images can be copied into the folder with their own data format.

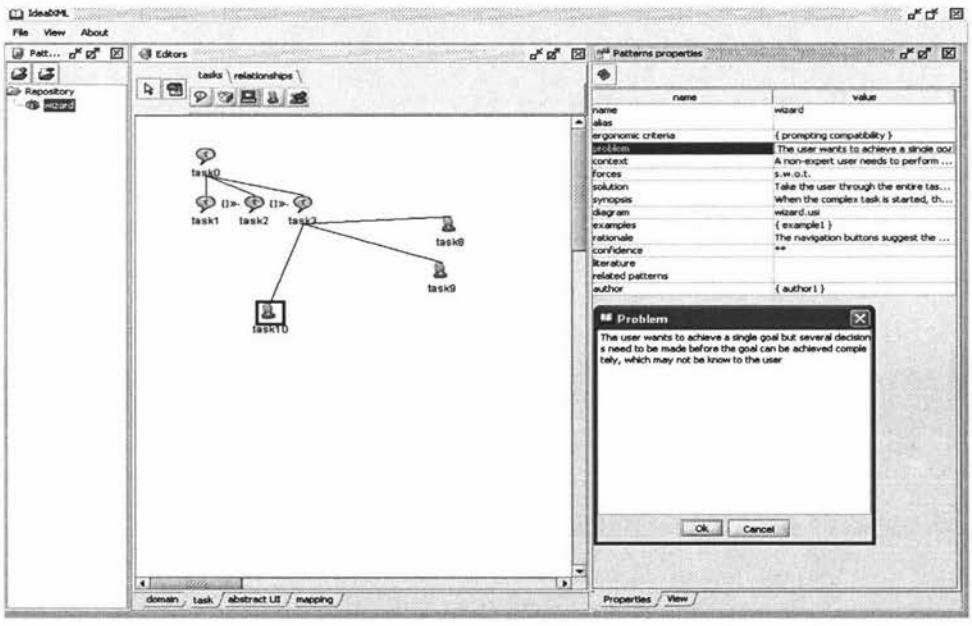


Figure 3.2 IdealXML.

All patterns manipulated by the system can be displayed in a list with a pattern name (see Figure 3.2). The detailed content of a selected pattern shows in the Pattern Properties window. The system names element fields of a pattern as properties and enables the user to edit data either in the editable field or in a pop-up window. The system allows the user to view the whole content of the selected pattern in a HTML page, which is similar to CoPE's HTML viewer.

The system supports a formal definition of mappings between models so that it is possible to add, remove and modify mappings during the development life cycle (Montero *et al.*, 2005). The models include domain, task, presentation, dialogue and context of use. The Editor contains four views for supporting the mappings: domain, task, abstract UI and mapping views. The domain view allows the user to draw the class diagram using UML notation. The task view presents the task model with CTT notation (Paternò *et al.*, 1997). The presentation model with abstract interaction notation can be designed in the abstract UI view. The above three notations can be mapped in the mapping view, which has not been sufficiently completed in the current version.

Based on the comparisons with other tools, the system emphasised less the relationships between patterns. It has not provided an efficient mechanism to link related patterns together, as it does not support the "pattern-link" element. There is no specific support for creating and naming relationships between patterns.

3.3.5 Summary of Existing Tools

Obviously, there is a progression in functionality and complexity from the pattern catalog tools to the pattern-based design tools.

The catalog-only tools normally give access to a small set of UI patterns presented using a format specific to the author. Importing existing patterns, creating new patterns and updating patterns are normally undertaken by the owner of the tool. Links between patterns and to other collections uses simple hyper-linking, which means that relationships cannot be named. Most use standard web browser facilities to search and browse their pattern collections.

Pattern management tools extend the functionalities found in catalog tools. All support pattern composition in a specific pattern format. Some are XML based but only one directly

supports PLML. Most provide specialist search facilities. MOUDIL offers a personal space for users to collect existing patterns so they can create a new personal collection. None of them support full pattern versioning and customisation. The majority of them support the creation of named relationships between patterns although the type of such relationships may be limited. With PET, CoPE and the IBM tool, the user can visualise the relationships between patterns. Manipulating forces gets minimal support from these tools. It is possible that both the CoPE and IBM tools should be categorised as pattern-based design tools, but without more evidence their present classification stands.

The pattern formats required by the pattern-based design tools are more specialised than those used when creating patterns for either the catalog or management tools. These tools help a designer find target patterns for solving a current UI design problem. Some of the pattern–based design tools support the cataloging tool functions and management functions, as well as functions for applying patterns to practical UI design problems. Both of the tools described by Henninger use a rule-based component to help guide the developer in selecting patterns based on their problem's characteristics. Damask and ONM can generate equivalent outputs for different target interfaces.

3.4 Conceptualisation of a Pattern Management Tool

The conceptualisation of a pattern management tool is based on the following principles: the need to support PLML, the needs of users, and the analysis of existing tools to identify their key features and shortcomings and further identify and categorise the key features for the proposed system. In this section, first the need to support PLML will be discussed. Second, the needs of users will be presented. Third, a comparison of existing tools will be conducted. Last, the conceptualization of the pattern management tool will be illustrated.

3.4.1 The Support of a Common Pattern Form - PLML

The proposed tool will manage pattern collections based on PLML. A common pattern form facilitates communication and cooperation between pattern tools. Patterns or pattern languages written in a common form are easy to refer to, whatever collection they are from. Such a standard form makes it convenient to identify common elements across collections and to identify the diversity among patterns. Therefore, patterns from disparate collections

can be gathered into a new collection for specific purposes. For example, a meta collection may be created for further extension by combining patterns from different collections.

3.4.2 Needs of Users

As mentioned in Chapter 1, potential users fall into four categories according to the main tasks they may wish to perform in relation to pattern collections: end users, UI researchers, UI designers, and UI developers. Different users need a variety of requirements in a tool to support their work. Due to the research scope and time constraints, the research focused on the needs of the UI researchers and designers who have more interest in UI patterns.

A UI pattern management tool can facilitate UI pattern related research activities for UI researchers. UI researchers need the tool to aid them to create new UI patterns based on the PLML template and to store them in a repository. They may want to correct or add a new version of an existing pattern without changing the original. UI researchers can then browse the various versions and analyse them, for example, to see the accumulation of evidence for the evolution of the pattern. Researchers also need the tool to peer-review existing pattern collections, to browse collections to gain an overview, and to add comments to individual patterns or pattern collections (Borchers, 2001). Researchers may also expect the tool to aid them in exploring the relationships between patterns or pattern collections.

UI designers need the tool to find out what to look out for in interactive exhibit UI design, to get a quick overview of an existing pattern or pattern collections based on the subject, and to find entries quickly via a keyword or text search (Borchers, 2001). The designers can use the tool to customise the patterns based on a current design problem and store it for future use. A small group of designers could work together on the tool and search for suitable pattern solutions for the sub-tasks, at first, and then integrate them together to solve the big design problem.

3.4.3 Proposed Key Features of the Tool with Analysis of Existing Tools

As a result of the analysis of specifications of existing pattern tools, it is possible to identify the required features for an ideal tool, a tool that meets the needs of users. The key features for such a tool can be divided into eight functional categories: pattern authoring, manipulating forces, browsing, searching, modification, relating patterns, manipulating

collections and input/output, which will be presented in detail below. The following descriptions analyses and summarises how existing tools support these features and outlines the shortcomings of these tools.

Pattern Authoring

Most of the tools provide different facilities to support the pattern creation process. All existing tools adopt a specific pattern form to represent UI patterns. Four of them can support alternative pattern structures. None of them provides a facility (input pattern template) to help the user map various pattern formats into the specific pattern form of the tool. The majority of systems can help pattern writers create a new pattern based on templates and some of them can support multimedia data, especially image data. Only one has an implementation code as a content field of the pattern.

Manipulating Forces

Forces amplify and illustrate the design problem being addressed by a pattern, as discussed in Chapter 2. Understanding forces may help researchers to better grasp how to construct a meta-pattern collection for a set of patterns. Therefore, the tool needs to support

- Browsing sets of forces to identify alternative patterns.
- Reusing forces when creating related patterns (Zajicek, 2004).
- Creating collections based on a set of forces (Zajicek, 2004).
- Selecting suitable patterns to solve a design problem.
- Analysing sets of forces to identify similarity and diversity between patterns within a collection.
- Exploring forces to identify possible links between patterns.

A few systems allow the user to edit the forces of a pattern. Browsing a set of forces allows the user to access the forces from the repository, which are already used by the existing patterns. Another objective is to aid UI designers to match appropriate or alternative patterns for the interface design. However, there is no tool to support this mechanism. Furthermore, none of them can identify and reuse existing forces set when creating new patterns or collections. The user can compare a set of forces between patterns or pattern collections so that it helps to understand the relationships between patterns or the hierarchy between collections.

Browsing Patterns

Browsing patterns is well supported by these existing prototypes. Almost all of them can display the patterns in lists, which allows the user to access and view the details of the content. Some can further allow the user to browse the same patterns in different views, such as in text view or graphical maps.

Searching Patterns

A searching facility is an important component for a pattern management tool. It has been implemented by many existing tools. There are five categories of search functions:

- *Keyword search* is to search target patterns based on keywords or relevant abbreviations.
- *Full text search* can find a pattern depending on any full text of any element content of the pattern.
- *Search based on different field* allows the user to search a specific pattern by searching the different element fields.
- *Search pattern based on any subset of fields* supports the user in finding the target pattern by searching the content of any combination of element fields.
- *Search pattern based on additional data* enables the user to filter patterns based on some specific data. For example, MOUDIL allows the user to search those patterns that include an implementation code or image data.

Pattern Modification

A UI pattern may evolve due to new technology and the depth of understanding of the pattern. Any change to a pattern should be annotated by the tool so that other users can know why the pattern was changed. The annotation is not only a change log but also allows the writing of review comments for further improvement. A number of tools from three different categories have supported this kind of mechanism.

Pattern versioning allows people to keep track of the evolution of a pattern over time thereby keeping a history of a pattern's development by recording modifications. The concept of pattern versioning is to keep track of various versions of a pattern within the system so that researchers can do a comparison between them and choose the one they wish to use. It also

enables the users to find a way back to the previous copies of a pattern if they make some mistakes in the current version. As a pattern matures its position within a collection's hierarchy may be more easily determined. Only the PoInter seems to partially support this function.

An existing pattern can be used as a template when creating a new but similar pattern. Using an existing pattern as a template is an efficient way to customise a pattern for either different audiences or for a specific domain. Such customisation of patterns should improve communication between participants during UI development (Lin & Landay, 2003). Pattern templates may also be used to tailor patterns according to special UI requirements, such as the three types of interfaces: web application, cell phone displays and voice interface (Lin & Landay, 2003). Storing the customised patterns enables them to be reused for future design problems. Damask can keep track of customised patterns for three different domains with different solutions.

Most existing pattern management tools provide the ability for the user to modify existing patterns, while the other two types of tools usually do not support this feature. This facility allows the user to change any elementary content of a pattern and save the modification. There are two methods used to rate the patterns from the tools surveyed: first, to rate patterns based on star scales in respect to how much confidence the pattern writer has in the patterns, appearing in Wu, PET, CoPE and IdealXML; second, MOUDIL rates either the top 10 or top 20 patterns, in terms of their popularity.

Relating Patterns

Individual patterns do not stand alone and the connectivity between patterns plays an important role in achieving a UI design that meets the design goals and objectives for a complex problem. A complex design problem may require a large number of inter-linked patterns to solve it. However, the relationship between patterns is often complicated. A mechanism is required to explore such relationships and discover the types of links that occur. Visualisation of overlapping hierarchies could facilitate this discovery process. Also such visualisations should help designers more easily understand the abstraction levels within a set of patterns and assist in using links when identifying which patterns are required to solve a specific UI design problem.

Providing support to facilitate the identification and creation process of links between patterns is another key activity of pattern related tools. In Chapter 2, it was pointed out that it requires a tool to aid researchers to explore the area of relationship types (Schuemmer, 2003b). Although CoPE and the IBM tool can create new relationship types, there is not a repository for accumulating various link types for future reuse or other research purposes. Three existing tools definitely allow the user to create relationships between related patterns, but only CoPE enables the user to name the relationship type. Four tools (Wu, PET, CoPE, IBM) have implemented the visualisation of relationships and displayed them in graphical maps, as mentioned above. Such a visual presentation can give the user a general intuitive concept of the relationships between patterns or collections.

Manipulating Collections

As disparate pattern collections contain a great many patterns, it is useful for identifiable pattern collections to be stored and organised in a management tool's repository thereby enabling:

- Easy access to existing pattern collections.
- Creation of new collections.
- Effective searching across collections.
- Easier comparison of patterns from different collections.
- Re-categorisation of patterns from disparate collections into larger composite collections.
- Using existing collections as templates for composing new collections that describe interfaces for specific domains (Gaffar *et al.*, 2003).
- Exploration and evaluation of the pattern collections.
- Exploration of relationships between patterns.
- Investigation of the associations linking patterns within and between the different collections.

Manipulating collections gets little support from existing tools. MOUDIL is the only one that allows the user to select a set of existing patterns from the repository for personal use. Manipulating multiple pattern collections within a repository can organise a large number of UI patterns in generic and logical ways. The mechanism enables re-categorisation of patterns from disparate collections into larger collections. But there is no clear evidence that indicates

whether or not the user can create new collections and categorise patterns into named collections.

Input/Output

There are different ways in which the systems can exchange pattern data with other resources:

- The system can import a single pattern at each time. A pattern can be written in a specific format and brought into the system. The following systems support this solution: PET and CoPE.
- The system enables the user to import a pattern collection set into the tool each time. The pattern set is written in a document with a predefined format. CoPE claims to import a collection of patterns in XML format.
- The system exports a single pattern into a target document. There are four tools that export a pattern into an XML document: PSP, PET, CoPE and IBM.
- The system can output a selected set of patterns. PSP allows the user to export any selected patterns into a XML document and CoPE supports exporting pattern collections either in XML or LaTex documents.

The pattern data format is various. A number of tools support XML data format like PSP, PET, CoPE, IBM, SWT and ONM. However, these tools do not support PLML in their current versions. Although IdealXML claims to support PLML, it does not support all of the elements from PLML. Some tools provide the output template to export patterns and three of them can support alternative output formats. For instance, CoPE can export patterns into diagrams, XML documents or LaTex documents and PET and the IBM tool allows different element fields in the pattern structure.

None of the existing tools support all the detailed features outlined in Table 3.3. Although most of them outline the importance of using a standardised format for patterns, none have as yet totally implemented PLML. PLML is required so that pattern data can be easily exchanged between different tools. As UI pattern tools need to deal with the variety of pattern structures, existing systems have not provided a facility to support mapping between the pattern format within the tool and any external resources' pattern formats. The power of forces has not generally been recognised as an aid for manipulating patterns, exploring the

relationships between patterns or selecting patterns to solve UI design problems. Most of the tools reviewed do not provide functions for creating new collections. None of them seem to support multiple versions of a pattern. There is little discussion about the provision of a mechanism for recognising new types of relationship, although Henninger's works use semantic web tools to investigate some of these issues (Henninger, 2001; Henninger *et al.*, 2003). Although CoPE allows for the creating and naming of relationship types, the system has no facility to accumulate the link types for future use or research. Furthermore, only Damask investigates the customisation of patterns based on the application domain to generate more illustrative examples.

3.4.4 Conceptualisation of a Pattern Management Tool

The proposed system is intended to manage a repository of possibly disparate pattern collections. The main features of the system are based on the existing specifications and tools. Two main groups of users, HCI researchers and UI designers, are intended to use the tool to create, manage and consume patterns. A comprehensive framework of features (Deng *et al.*, 2005) has been proposed. The proposed framework for managing collections of patterns includes specific features identified from the analysis of both the existing specifications and the tools. Table 3.3 presents the framework for a pattern management tool with the features required listed in columns one and two. The features of the framework have been grouped into eight functional categories:

- *Pattern authoring* supports activities associated with the creation of patterns using a flexible template based on PLML (in which many of the fields are optional).
- *Manipulating forces* allows new forces to be created. These are added to the forces list. This list can be browsed when creating or modifying patterns. Forces can be used to explore relationships within pattern collections. They can also be used for selecting patterns when building a solution for a specific UI design problem.
- *Browsing* supports accessing patterns in different formats including displaying the full detailed content of a selected pattern.
- *Searching* provides keyword, full text and advanced search facilities. An advanced search can find patterns based on a set of fields from the pattern format or by selecting from dynamically generated lists.
- *Modification* supports pattern versioning and customisation as well as annotations attached to changes including the rating of patterns.

- *Relating patterns* supports the recognition of new types of relationships, creation of relationships and the visualisation of the relationships.
- *Manipulating collections* provides facilities for re-categorising patterns into different collections as well as for creating new collections.
- *Input and output* allows for the import and export of single patterns or whole pattern collections in alternative formats.

Table 3.3 Features for a pattern management tool

Features \ Tools		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Authoring	Specific pattern form	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
	Alternative pattern form	Y				Y					Y	Y					
	Input pattern template																
	Create new pattern using template					Y	Y	Y	?	?	Y	Y	Y	Y		?	Y
	Support multimedia data	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y		Y			Y
	Include implementation code								Y								
Manipulating forces	Create force						?	?	Y	?	?	Y		Y			Y
	Browse a set of forces																
	Identify/reuse forces																
Browsing	Display pattern list	Y	Y	Y	Y	Y	Y	Y	Y		Y	Y	Y	Y	Y	Y	Y
	View pattern details	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y	?	Y
	Different views						Y		?	Y	Y	Y					Y
Searching	Keyword search					?		Y	Y	Y		Y					
	Full text search					Y		Y	Y	Y		Y					
	Search based on different fields							Y	Y	?		Y					
	Search pattern based on any subset of fields								Y			Y					
	Search pattern based on additional data					Y		Y	Y			Y				?	
Modification (Versioning & customizing)	Annotate	Y				Y		Y	Y	Y		Y			Y	Y	
	Pattern Versioning					?											
	Keep track of customisation of patterns					?								Y			
	Modify patterns					Y		Y	?		Y	Y					Y
	Rating of patterns						Y		Y	Y	Y						Y
Relating patterns	Create relationship type										Y	Y					
	Add relationships								?	Y	Y	Y				?	
	Naming relationships										Y						
	Visualise relationships					?	Y			Y	Y	Y				?	
Manipulating collections	Collect a set of patterns for personal use								Y								
	Categorise patterns into named collections								?								
	Create new collections								?								
Input/output	Import single pattern									Y	Y	?					Y
	Import a collection of patterns									?	Y	?					
	Export single pattern							Y		Y	Y	Y					Y
	Export a collection of patterns							Y			Y						
	Export pattern map									?	Y	?					
	Support XML							Y		Y	Y	Y			Y	Y	Y
	Support PLML																Y
	Provide output template							Y		Y	Y	Y					
	Support alternative output formats									Y	Y	Y					

Legend

Y: The tool supports the feature ?: Feature partially supported

1: PID **2**: CG **3**: UIT **4**: UIDP **5**: PoInter **6**: Wu **7**: PSP **8**: MOUDIL
9: PET **10**: CoPE **11**: IBM **12**: GUIDE **13**: Damask **14**: SWT **15**: ONM **16**.IdealXML

The details of features from each category have been outlined in Section 3.4.3. A comprehensive pattern management tool to support pattern collection management has to cover the features described in Table 3.3. The conceptualised system should improve the shortcomings of existing tools and manipulate the pattern contents based on the standard PLML format.

3.5 Summary

This chapter has surveyed the existing specifications and tools used to support UI pattern management systems. First, the existing specifications for building a pattern management tool have been described individually and common features summarised. Second, a selection of UI pattern tools have been reviewed and categorised into three types based on the type of definitions. These computer systems have been compared with each other to identify common functional features. Third, a conceptual feature framework has been introduced based on the previous sections' discussion and the needs of users. The shortcomings of existing systems will be improved by the proposed system. The next chapter will explore the details of the prototype.

Chapter 4 Prototype for MUIP

This chapter describes the prototyping and implementation of a pattern management system (MUIP) for managing UI pattern collections. First the specification of the system is introduced, along with the new version of PLML, as the standard pattern form. Secondly, the design issues relating to the system architecture, the database design, the development tools and the prototyping are described. Thirdly, the prototyping of key aspects of MUIP is discussed and illustrated with appropriate examples. Finally, the extent to which the prototype meets the requirements specified is considered.

4.1 Feature Specification

As discussed in Chapter 3, MUIP focuses on the needs of UI researchers and UI designers. MUIP is proposed to manage a repository of disparate pattern collections to support the potential needs of these two kinds of users. Therefore, MUIP needs to support the following main activities:

- Enable the user to compose patterns based on PLML.
- Aid the user to explore the power of forces and manipulate forces within the system.
- Provide searching and browsing facilities to search and view target patterns or collections.
- Support pattern modification including pattern versioning and customisation.
- Link patterns and accumulate the types of relationship.
- Manipulate pattern collections.
- Import and export patterns in various formats.

In this section, the detailed requirements of MUIP emerging from the task analysis are summarised. Then, since one of the above main features is that the system is based on PLML, Section 4.1.2 introduces the related work on the evolution of the PLML v1.1 and presents the new version, PLML v1.2.

4.1.1 Analysis

Use cases describe the interaction between a user and the system (Fowler & Scott, 1997). Therefore, use cases can model the actions of the users and the performance of a system. A use case diagram is a useful tool to illustrate the requirements of a system. The diagram addresses all the use cases of a system and the relationships between the actors and the use cases. However, the use case diagram does not present the detailed content of each use case, so it requires another form to present them in detail, such as use case descriptive scenarios.

According to the features outlined in Chapter 3 and the objectives of the system, three kinds of actors have been identified: UI researchers, UI designers, and database administrators. Both UI researchers and designers interact with the system by dealing with the pattern collections data.

The MUIP system aims to allow users to manage and consume UI patterns. Besides basic pattern authoring, the researchers may want to categorise patterns and identify their relationships. Pattern collections may be reviewed and act as templates for creating larger collections. UI designers may wish to browse existing patterns from the repository to match an appropriate set of patterns to their current design. A group of designers may use the system to work together to find out a solution for a complex design problem. The description below summarises the interactive activities that UI researchers and designers may perform in the system:

- Registering as the user of the system and login to the system;
- Searching for existing patterns based on keywords, full text, different elementary contents, or a set of different fields of pattern elements;
- Creating a new pattern based on the PLML v1.2 template. Most of the elementary fields are optional so that the user can map the template with existing pattern forms;
- Browsing patterns from the different pattern group lists, viewing the content of a selected pattern, and editing review comments;
- Manipulating forces, including creating, browsing, searching, reusing, and labelling forces. MUIP enables the browsing of a set of forces of a pattern or a collection and allows the user to find patterns based on forces. The creation of a new pattern or collection can reuse existing forces.
- Modifying and updating the content of patterns. Pattern writers can comment on changes, but some critical changes should be automatically logged, such as the

change of name and version. MUIP should support the versioning of pattern and pattern customisation;

- Identifying the relationship between patterns and linking patterns or collections. MUIP can create and accumulate new types of relationships and then apply the relationships to connect patterns or pattern collections. MUIP will enable the user to create the visualised relationship map;
- Manipulating pattern collections, including the creation of collections and pattern categorisation. The facility allows the user to create new collections and re-categorise existing patterns into the variety of categories based on the named collections, and browse patterns from each collection;
- Importing and exporting patterns using the PLML format or alternative formats. A pattern or a pattern collection may be imported into the system. The pattern written in PLML v1.2 should be imported automatically, including image data. The system provides a facility to import patterns in other formats manually, such as HTML, RTF, Word DOC, or PLML v1.1, etc. The system can export patterns or pattern collections into alternative formats, such as PLML v1.1, PLML v1.2 or RTF. An advanced facility allows the user to import and export patterns based on their own predefined XML schema.

The database administrator maintains the pattern repository, including the insertion, update and deletion of patterns or pattern collections. Figure 4.1 shows the use case diagram which illustrates the actors and the connections with use cases that the system supports. The use case diagram is the overall model of the system. The detailed specifications of these use cases and some major scenarios are presented in Appendix C.

Table 4.1 summarises the mappings between the main features (see Chapter 2) and the use cases:

Table 4.1 Mappings between the main features and the use cases

Main Features	Use Cases
Authoring	Create new pattern
Manipulating forces	Manipulate forces
Browsing patterns	Browse pattern, Write comment
Searching patterns	Find existing pattern
Modification	Modify pattern, Create new version, Customise pattern, Write comment
Relating patterns	Link pattern, Create link type
Manipulating collections	Manipulate collection
Input/output	Import pattern, Import PLML, Import Alternatively Formatted Pattern, Export PLML vx.x, Export Pattern in Alternative Format

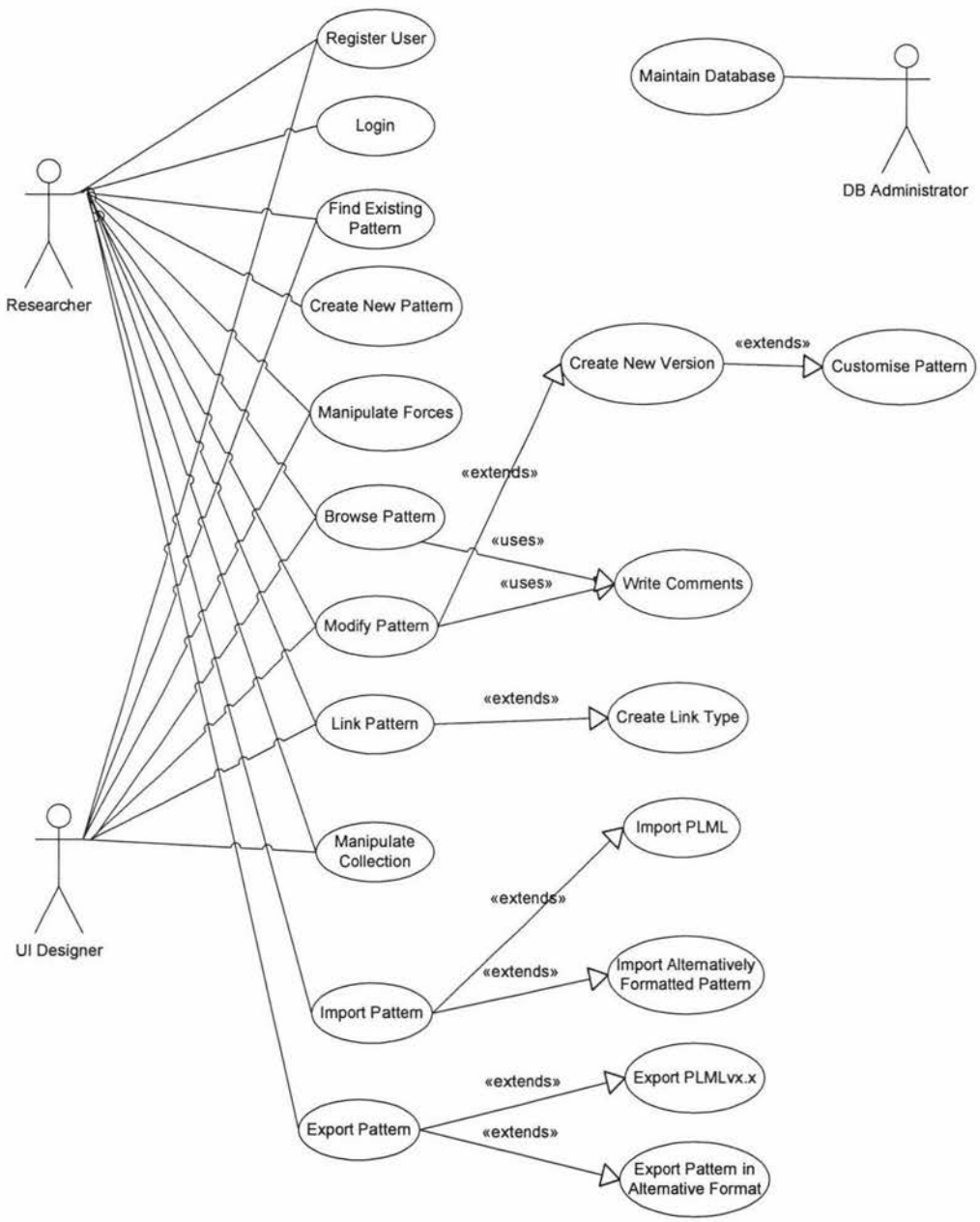


Figure 4.1 Use case diagram for MUIP.

4.1.2 PLML v1.2

After PLMLv1.1 was firstly introduced in the CHI2003 pattern workshop (Fincher, 2003), some researchers continued to explore ways of improving PLML v1.1. An Extended Pattern Language Markup Language (PLMLx) was developed by Bienhauas (2004). Compared to PLML v1.1, the main changes in PLMLx included:

- Element "name", "problem", "context", "forces" and "solution" are set as mandatory elements;
- Changing element "example" and "rationale" as first level elements which act as the children of element "pattern";
- Adding an element "ptname" so that pattern author can define a special font effect for pattern names, like small capitals;
- Adding elements "copyright", "license", "change-log" into the element "management";
- Element "change-log" consists of a sequence of "changes". Each "change" can be further characterised by a "version", one or more "authors" and a "description". The element "version" can contain a major ("majorNr") or minor ("minorNr") release number and "date" elements;
- Element "last-modified" is deleted because of the "date" element within element "version";
- Adding elements "acknowledgments", "resulting-context" and "organization";
- The element "organization" consists of sub-elements "category", "classification" and "collection".

Although PLMLx made improvements to the original specification, some researchers have not agreed with some of the suggested changes. Fincher (2004) argued that making the elements "problem", "context", "forces" and "solution" mandatory will cause PLMLx to be less useful. She further disagreed with bringing the elements "rationale" and "example" to the first level because this will destroy their original function in the pattern form. The element "resulting context" was problematic and could be easily stated in the element "related patterns". PLMLx has also reordered some elements and separated "context" and "forces" into non-adjacent positions, which is not consistent with the Alexandrian and other pattern forms.

We have refined and extended the current PLML v1.1 to a new version PLML v1.2, with more sub-elements, rather than reorder the elements (see Appendix B). The key reasons for these modifications are presented as follows.

First, in PLML v1.1, only the content contained inside separate tags is searchable and accessible. Therefore, PLML v1.1 needs to refine those elements containing different multiple objects. For example, each pattern may have numerous different forces and a separate force is not allowed to be accessed individually in the PLML v1.1 definition. PLML v1.2 contains forces separately and makes them individually accessible for the purpose of searching and reusing individual forces.

Second, it is necessary to identify different objects with various types of data inside elements. Evidence for patterns can have many examples which include known uses (Fincher, 2003, p.27). Each example may have a narrative description and illustrations, such as pictures. PLML v1.1 does not allow numerous examples with the narrative descriptions and binary image information to be placed inside one element in a XML file. To identify different objects, authors have appended sub-elements for those elements, such as example, literature, and implementation. Those sub-elements originate from Fincher's discussion (2003) and knowledge from current known patterns.

Third, each pattern may have different versions after modifications have been made. Therefore, PLML v1.2 adds a change-log for patterns because it is necessary to know why and how a pattern is changed. The change-log is organised by date according to temporal-based modification activities. The element pattern-link is appended with the revision-number attribute that enables the creation of the relationships between different versions of patterns.

Table 4.2 illustrates the main differences of the two versions' DTDs. In summary, compared to PLML v1.1, PLML v1.2 can construct and organise pattern contents more effectively. It has further defined the structure with details for forces, example, literature, and implementation, as well as additional change-log data. A comparison with PLMLx indicates that PLML v1.2 is closer to PLML v1.1, so it is easier to map with PLML v1.1. Both PLML v1.2 and PLMLx add the element "change-log" and address low level elements. However, PLML v1.2 and PLML v1.1 do not include information on categories or classification, while PLMLx contains this kind of data.

Table 4.2 Comparison between PLML v1.1 & PLML v.1.2

Versions / Elements	PLML v1.1	PLML v1.2
Attributes for Pattern	<!ATTLIST pattern patternID CDATA #REQUIRED >	<!ATTLIST pattern patternID CDATA #REQUIRED collection CDATA #REQUIRED>
Element "forces"	<!ELEMENT forces ANY>	<!ELEMENT forces (force *)> <!ELEMENT force ANY>
Element "example"	<!ELEMENT example ANY>	<!ELEMENT example (example-name?, example-diagram?, description?, knownuses?)> <!ELEMENT example-name (#PCDATA)> <!ELEMENT example-diagram ANY> <!ELEMENT description ANY> <!ELEMENT knownuses ANY>
Element "literature"	<!ELEMENT literature ANY>	<!ELEMENT literature (workname?, reference?)> <!ELEMENT workname (#PCDATA)> <!ELEMENT reference (#PCDATA)>
Element "implementation"	<!ELEMENT implementation ANY>	<!ELEMENT implementation (implementation-name?, code?, otherdetails?)> <!ELEMENT implementation-name (#PCDATA)> <!ELEMENT code ANY> <!ELEMENT otherdetails ANY>
Element "pattern-link"	<!ATTLIST pattern-link type CDATA #REQUIRED patternID CDATA #REQUIRED collection CDATA #REQUIRED label CDATA #REQUIRED>	<!ATTLIST pattern-link type CDATA #REQUIRED patternID CDATA #REQUIRED collection CDATA #REQUIRED revision-number CDATA #REQUIRED label CDATA #REQUIRED>
Element "management"	<!ELEMENT management (author?, credits?, creation-date?, last-modified?, revision-number?)>	<!ELEMENT management (author?, credits?, creation-date?, last-modified?, revision-number?, change-log*)>
Element "change-log"		<!ELEMENT change-log (log-creation-date?, log-content?)> <!ELEMENT log-creation-date (#PCDATA)> <!ELEMENT log-content ANY>

4.2 Design Issues

4.2.1 System Architecture

The MUIP system employs the Client-Server architecture described below. The system allows the individual user or a group of users to access a central repository to manipulate the UI pattern collections. A researcher can compose and store new pattern collections into the networked central repository and share these with other researchers, for example, they can access and review these pattern collections. The Client-Server architecture also enables a group of UI designers to work together and search patterns for solving their design problems.

The Client-Server computing system generally consists of two major logical parts: the server supplies the services; and the client requests the service from the server for his/her own computing purpose (Lewandowski, 1998). In this paradigm, single or multiple clients and one or more servers, depending on the underlying operation system and communicating

systems, compose an integrated system providing services for distributed computation or other purposes (Sinha, 1992).

A server of a Client-Server system provides services to the clients. The services of Client-Server systems operate according to their own goals and objectives. The server usually waits for the requests for services from the clients, but recent changes allow the server to automatically detect the status of the client and response with appropriate actions (Lewandowski, 1998). The server can encapsulate the provided service and hide the status of the server from the end-user and clients so that the clients should not need to be concerned with the implementation issue of the backend server, such as the platform.

There is a middle tier (server interface) to facilitate the internal communication between the clients and the server. The middle tier provides the encapsulated services for the clients to retrieve data for their analysis. It also includes a variety of components such as the underlying communication system, and some standard database connection component like ODBC (Lewandowski, 1998). Figure 4.2 illustrates a model of Client-Server architecture. A database server provides the data service to multiple clients and a number of clients request services from the server, along with the underlying interprocess communication systems (server interface) (Sinha, 1992).

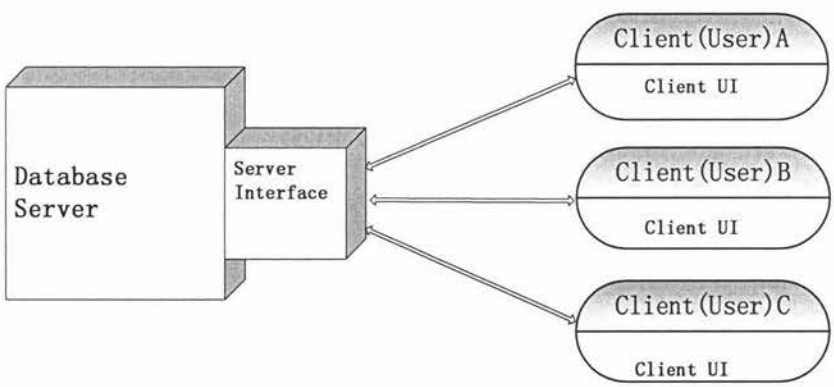


Figure 4.2 A model of Client-Server system, adapted from Lewandowski (1998).

The clients request the services from the server by using query or command languages, which can then be interpreted by the server. A Client-Server may use the Structured Query Language (SQL), or other specific database languages. SQL is the first and most widely accepted standard database language, supported by more than one hundred Database

Management Systems (DBMS), running on various operating system platforms and hardware platforms (Connolly & Begg, 2004).

The development of the MUIP system can be logically divided into four tiers: presentation, the logic of manipulating UI pattern collections, data access and data storage, and management. Figure 4.3 illustrates these layers (adapted from Lhotka, 2003). The presentation layer includes all the user interfaces of the system, presented in windows form. These user interfaces will be discussed in the prototyping section below. The logic of manipulating UI pattern collections is closely related to the presentation layer. How to manipulate UI pattern collections can found in the feature specification section. This logic has been implemented in the source code based on the predefined user interface. The data access layer provides the facility for the presentation layer to connect with the data storage layer. This layer employs and implements many functions from the server interface tier of the Client-server architecture, as illustrated in Figure 4.2. The database and management layer describes the backend database structure based on a DBMS for storing UI pattern collections. The details of the design are discussed in Section 4.2.2.

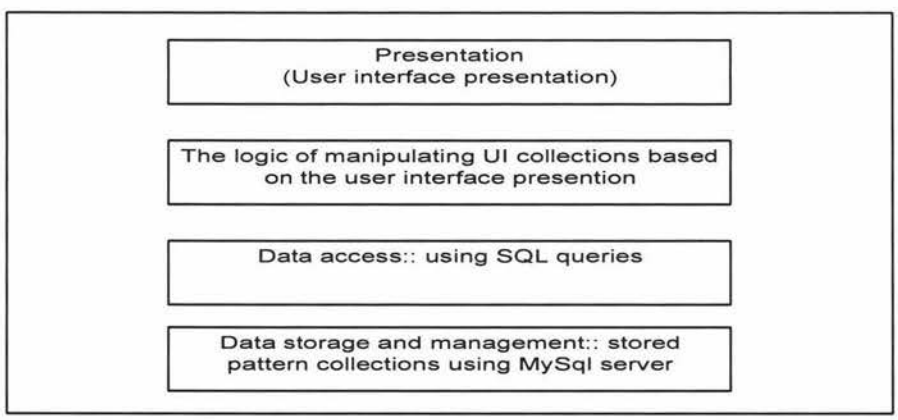


Figure 4.3 Different tiers of the system, adapted from Lhotka (2003).

4.2.2 Database Design

The MUIP system is a pattern-oriented system, so most of the data the system deals with is pattern-related. The pattern structure of the tool is based on PLML v1.2. The elementary data of PLML v1.2 should be supported by the backend database. The database should hold data about users as well as patterns. Thus the database will:

- Store the text or string data regarding the descriptions of patterns based on PLML v1.2;

- Use numeric data to present the Pattern ID;
- Include PLML date time data;
- Include media data such as image data to illustrate patterns or address examples;
- Record the information about pattern collections;
- Keep track of various versions of patterns;
- Accumulate the types of relationships and store the relationships between patterns;
- Store a set of forces;
- Support registered users' information;
- Support standard SQL.

Database design consists of three major phases which include the conceptual database design, logical database design and physical database design (Connolly & Begg, 2004). The MUIP system uses the Entity-Relationship (ER) model to aid conceptual and logical database design. The ER model is a conceptual model with a top down approach to aid the development of database structure design (Connolly & Begg, 2004). The ER model identifies the most important data names as the "entity" and a database can includes many different entities. The associations between the entities are called "relationships" within ER modelling. The "attributes" are the detailed information used to describe the entities or relationships. The ER diagram provides an efficient method to visualise ER modelling. This technique has been proven useful in aiding database designers to translate the database requirement specifications into a formal and logical database model.

Figure 4.4 illustrates the ER model of the system database, which identifies the entities with their attributes and relationships. The entity "Patterns" can contain the following entities: Example, Literature, Log, Management, Implementation, Forces, Alias and RelatedPatterns. The Collection entity can include many different patterns. The RelatedPatterns entity can contain various relationship types. The OurUser entity can create the relationship types and manage the patterns or collections.

The first step of physical database design is to grasp the information on the relationships illustrated in the ER diagram (logical model). The information on each relationship for the logical model can be obtained by using the Database Design Language (DBDL) that can consist of (Connolly & Begg, 2004, p.498):

- The name of the relation;
- A list of simple attributes in brackets;

- The primary key and, where appropriate, alternative keys (AK) and foreign keys (FK);
- The integrity constraints for any foreign keys identified.

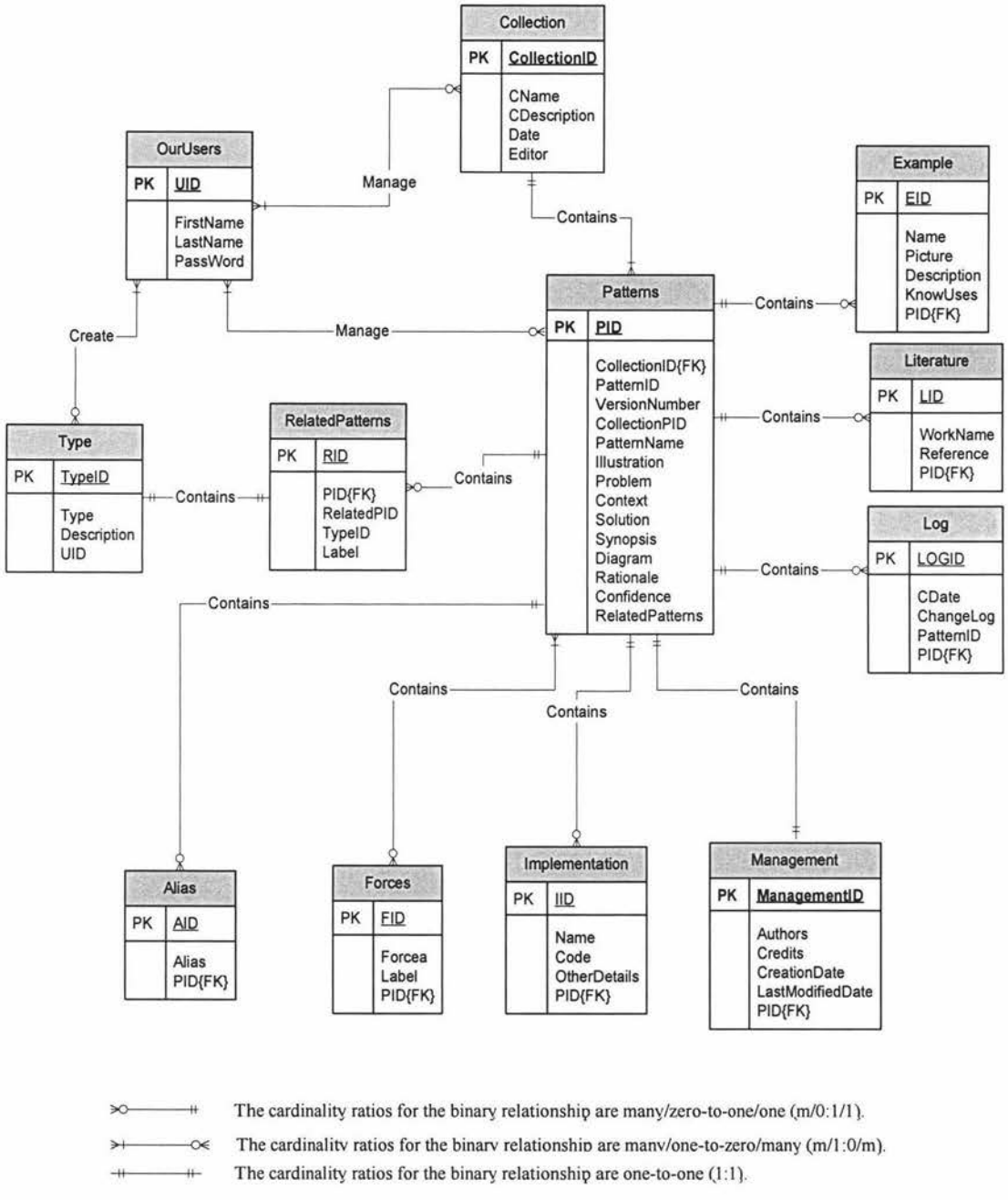


Figure 4.4 ER model of the database.

Figure 4.5 is an example of the DBDL for the relation of the OurUser case study:

```
OurUsers (
        UID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        FirstName VARCHAR (50) NOT NULL ,
        LastName VARCHAR (50) NOT NULL ,
        PassWord VARCHAR (50) NULL ,
        PRIMARY KEY(UID)
    )
```

Figure 4.5 DBDL for setting up the OurUser relation.

The next step is to implement the base relations into MySQL. A collection of information in a relational database of MySQL is organised into a set of tables. The process of physical database design is to convert the base relations into a real database schema for producing a relational database in MySQL. The database schema of MUIP was created in a data file named muip.sql, which can be found in the Appendix D. The database structure is the same as the ER diagram described in Figure 4.3. In the muip.sql file, the relation of OurUser was implemented in SQL statements and this is illustrated in Figure 4.6:

```
CREATE TABLE OurUsers (
        UID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        FirstName VARCHAR (50) NOT NULL ,
        LastName VARCHAR (50) NOT NULL ,
        PassWord VARCHAR (50) NULL ,
        PRIMARY KEY(UID)
        );
```

Figure 4.6 SQL for creating the table OurUsers.

For the security of the database system, the database server of MUIP is running as a non-root user that requires the setting up of a user account and access privileges. The database will create a specific user account and grant privileges to insert, update, delete, select, drop and create, while the database generates based on muip.sql.

4.2.3 Development Tools

The system was implemented using Microsoft C# as the development language, Microsoft .NET Framework, and the relational DBMS MySQL.

The selection of the programming language was based on the following considerations. C# language is a standard computer programming language which is widely accepted and used to develop business applications (Microsoft). The Microsoft .NET Framework supports language interoperability so that C# has the ability to work with existing source code, written in different programming languages such as VB, J#, C++, and J++ (Mok, 2003). Similar to

Java, C# is designed to be a platform independent language based on the .NET Common Language Runtime (.NET CLR) so that the program can run on various operating systems, like Windows and Linux. Similar to Java, C# is Microsoft's new object-oriented language, specifically designed to take full advantage of the .NET type system. C# provides the APIs that cover almost every aspect of development such as the GUI design and database access (ADO.NET). The Microsoft .NET Integrated Development Environment (IDE) enables the designer to create the GUI efficiently. Most major databases provide the APIs to support the C# database access function. Furthermore, the Microsoft .NET framework is free and available to download from the Internet.

A relational database was required to store the pattern collections because of the relational logical database model illustrated in Section 4.2.2. MySQL is a relational DBMS, which was selected for this project based on the following features (DuBois, 2000):

- MySQL provides the fastest speed and high performance compared to a set of popular databases.
- MySQL is an open source database, and it is easy to obtain a copy and simple to install.
- MySQL supports the standard SQL and enables the clients' applications to access the database server by using ODBC (Open Database Source Connectivity), a database communication protocol developed by Microsoft. The Microsoft .NET product can use the ODBC class library of ADO.NET to request service from the MySQL database server.
- MySQL allows multiple users to connect to the server simultaneously.
- MySQL is a network-based database that can be accessed from the Internet or a local network. MySQL has the access control to set privileges for legal database users.
- MySQL can run on a variety of operating systems such as UNIX, Windows and OS/2.
- MySQL can import and export a database with data information easily.

The list below summarises the detailed software packages involved in the development of the MUIP system. The Microsoft Visual Studio .NET 2002 is an integrated development environment (IDE) that enables software developers to design software applications in C# or other programming languages. All the other software packages are required when MUIP is running on a computer. The list includes:

- Microsoft .NET Framework SDK 1.0;

- Microsoft ODBC .NET, an add-on component as data provider to the .NET Framework;
- Microsoft Data Access Components (MDAC) 2.8, containing the core data access components such as ODBC driver;
- MySQL database v4.1;
- MySQL ODBC 3.51 Driver, providing access to the MySQL database system; and
- Microsoft Visual Studio.NET 2002.

4.2.4 Prototyping

There are two major types of prototyping: low-fidelity prototyping and high-fidelity prototyping. Low-fidelity prototyping is characterised as the simple and cheap method to translate high level design concepts into a low-fidelity prototype, including storyboarding and sketching (Preece *et al.*, 2002). High-fidelity prototyping creates a prototype that is similar to the final product. For instance, the designer can use Visual Basic to produce a high-fidelity prototype rather than a paper-based artefact.

The prototype of MUIP adopted high-fidelity prototyping because it has several advantages (Preece *et al.*, 2002):

- It can complete full functionality;
- It is fully interactive;
- It is a user-driven method so that the user can review the prototype directly;
- The navigational scheme has been clearly defined;
- The prototype can be used for exploration and testing;
- It looks close to the final product;
- It serves as the living specification for further improvement.

As the prototype of MUIP was built on software, a prototyping tool will facilitate the creation process. User interface prototyping and IDEs have played an important role in the software development process (Myers, 1995). These help UI designers reduce the work of coding when producing user interfaces efficiently. Myers (1995) states that an IDE can produce consistent user interfaces of good quality, and the code for generated the interfaces is easier and more economical to both create and maintain.

MUIP was developed using the Microsoft Visual Studio .NET 2002 IDE as both the prototyping and the development tool. This IDE provides a similar prototyping component to support the user interface design across programming languages such as VB and C#. The user interface can be designed easily by dragging and dropping the target interface elements into the proper position on the form. It is also easier for UI designers to modify the resulting prototype.

Developing the prototype

The class diagram shown in Figure 4.7 models the structure of the program. There are seven user interface classes: MUIPMainUI, UserRegisterUI, LoginUI, PatternContentUI, ForcesUI, RelationshipTypeUI, and CollectionUI. These interfaces were developed around the activities described in the use case specifications in previous section. They implement the first two architectural layers: the presentation layer and the logic layer for manipulating the UI collections based on the user interface presentation, illustrated on the Figure 4.3. The detailed functions of these interfaces will be illustrated in the following sections. The MUIPData class handles interaction with all pattern data from the backend database. It provides a wrapper for code that accesses the data for the given application. This ensures that the database connections are not exposed externally to the rest of the system. The MUIPData class provides the functions for the data access layer. All user interface classes have a dependency link to the MUIPData class because these classes all need to use the data access functions to get related data and display data in the user interface components. The PatternDataSet class provides a consistent data structure which contains pattern contents so as to facilitate data exchange between the classes. As the Microsoft .NET framework and MySql database server support different date-time formats, the class MUIPDateTime provides methods to map the .NET framework date-time format to the MySql database server date-time format. The PLMLHandler class parses all pattern contents in XML format and converts them to PLML format; it also provides functionality to export pattern data into PLML. The Users class provides containers for detailing user information.

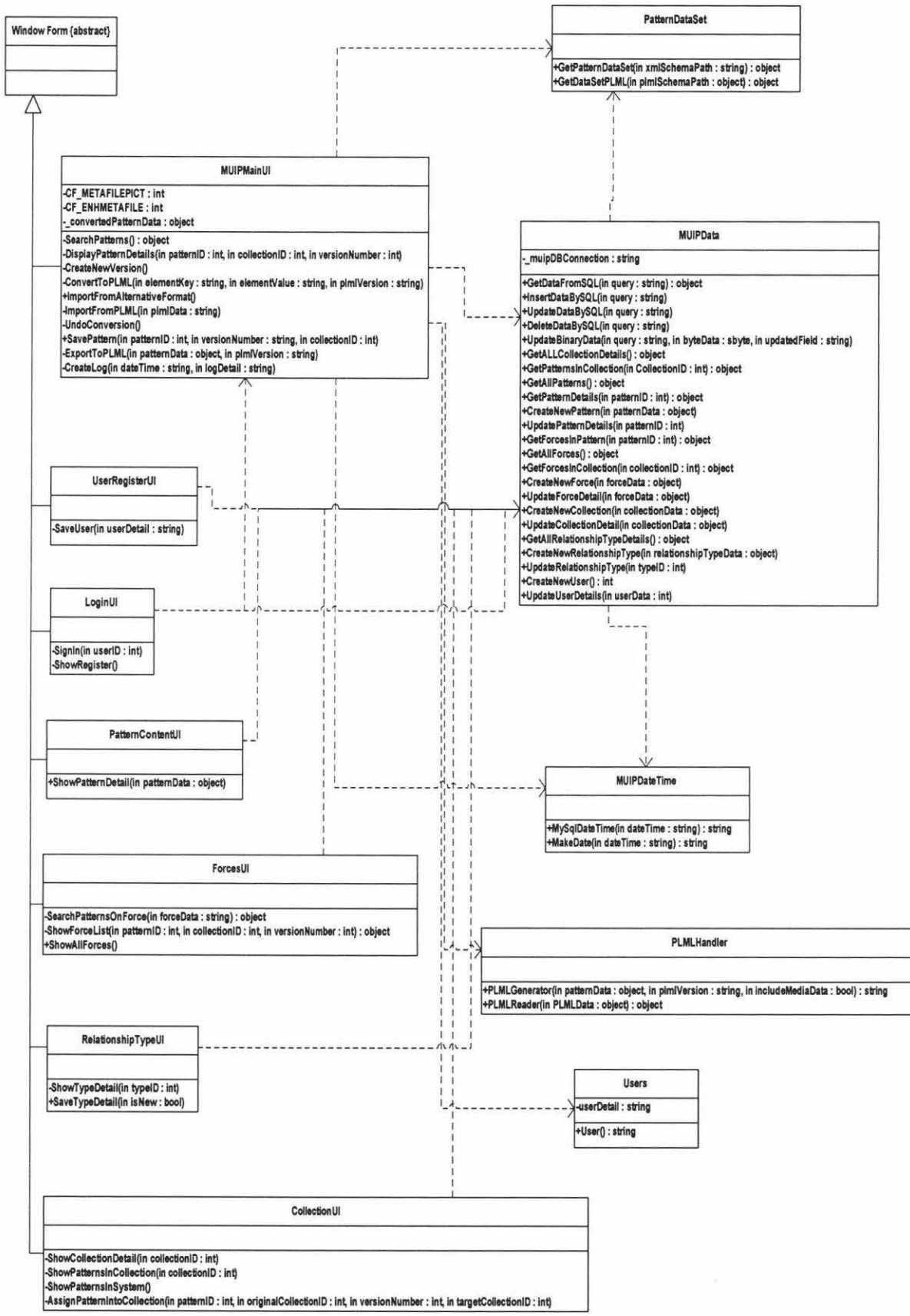


Figure 4.7 Proposed Class Diagram of MUIP.

The main interface of the system

The main interface of the MUIP system includes two major components: the pattern editor and the pattern finder, as shown in Figure 4.8. The pattern editor uses the PLML v1.2 template so the user can edit and view a UI pattern. The pattern finder function provides fields so the user can search for existing patterns in the repository.

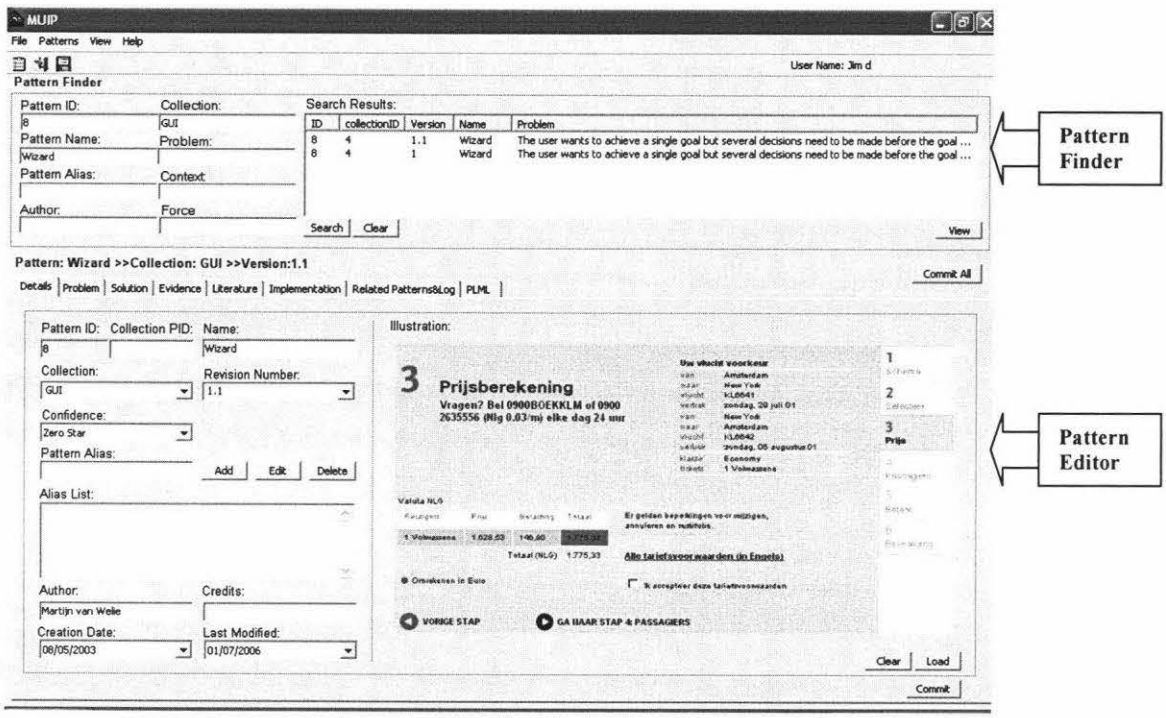


Figure 4.8 Proposed main interface of MUIP.

Authoring

The pattern editor has a tab-based interface. Figure 4.9 shows all eight tabs. The elements of PLML v1.2 are grouped and presented under these tabs and effectively represent the template. The user can navigate the tabs to enter the pattern content into appropriate fields when creating a new pattern. Most of these elementary fields are optional so that the user can select the necessary elements to map from existing patterns that use alternative structures. However, a new UI pattern must have at least a name, collection, revision number and a pattern ID (generated by the system because a pattern must have a unique pattern ID) but it may also have different versions and belong to various collections. The tool supports adding and storing image data into the fields: illustration, example-diagram and diagram. Multiple implementations in the form of source code can also be added to a pattern using the editor.

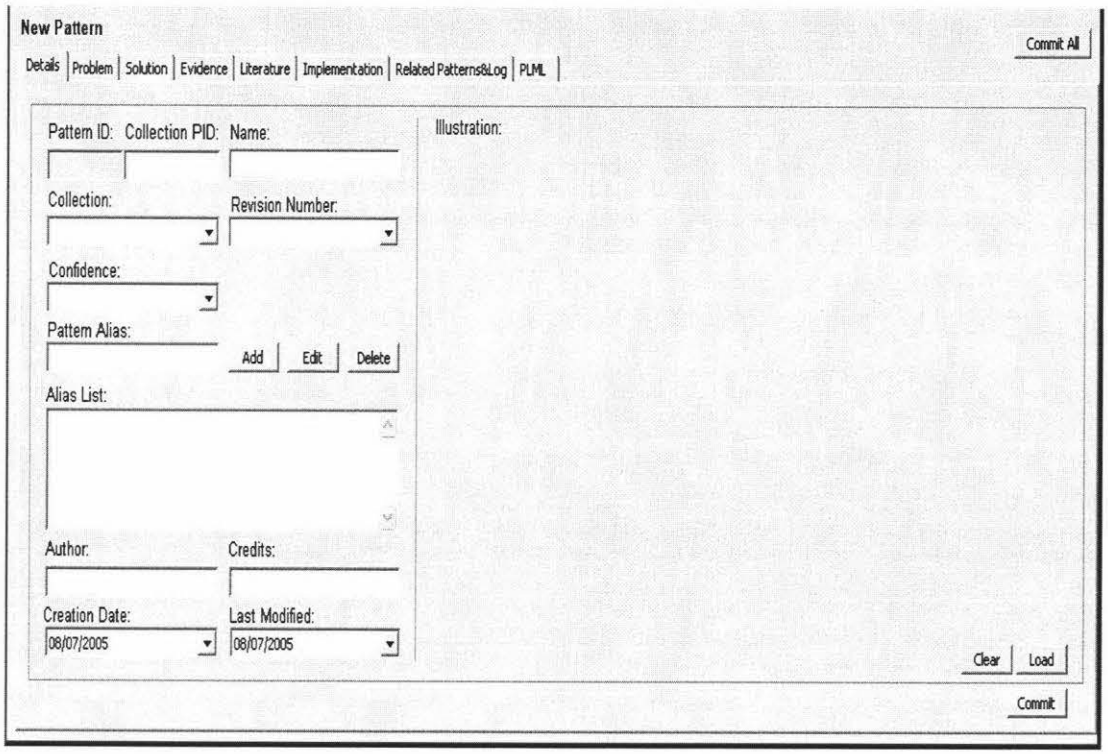


Figure 4.9 Pattern Editor.

Manipulating forces

The user can create forces and store them independently from patterns. Each force may have a label or other details added as an additional explanation. The set of existing forces can be browsed to view their details. The user can then select a force and search for any related patterns which contain the same force or a similar force. The forces attached to any of the patterns in the search set can be further browsed by selecting the target pattern in the *Search Results* list. The set of forces for a whole collection can be viewed in the *Force List* by selecting the target *Collection* name from the collection list. The screen layout for these functions is presented in Figure 4.10.

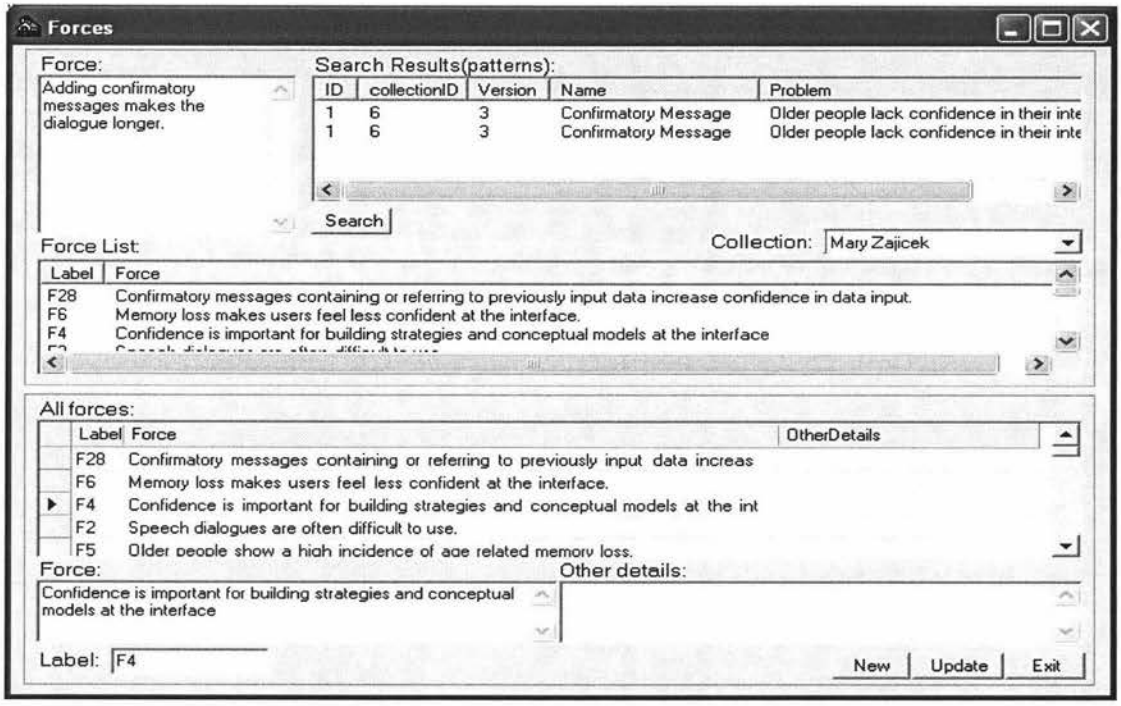


Figure 4.10 Forces View.

In the forces field of the pattern editor (see Figure 4.11), the forces belonging to a pattern are displayed in the force list. The force list allows the user to view, add, update and delete forces. There is a search facility to help the user identify and reuse any existing forces from the repository while creating the force list for the new pattern or when updating the force list for a pattern. This function can search forces based on the force label or any keyword appearing in the text of a force.

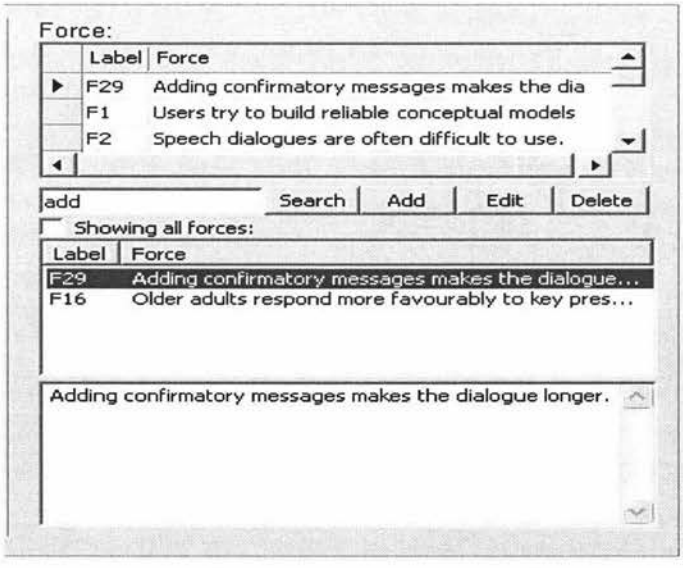


Figure 4.11 Forces field of Pattern Editor.

Browsing

Figure 4.8 shows the details of the selected pattern displayed in the pattern editor, when a user selects a pattern from the *Search Results* list in the pattern finder. The user can navigate the different tabs to browse the contents of the pattern. An alternative way to display the pattern is by right clicking the mouse on the selected pattern in the *Search Results* list, this will open up the pattern wizard as shown in Figure 4.12. This is the pattern content view of the selected pattern.

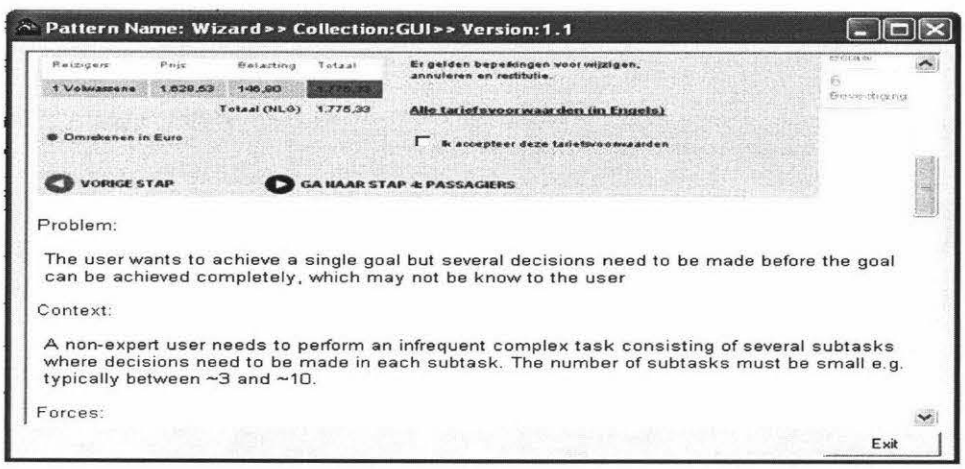


Figure 4.12 Pattern Content View.

Searching

Figure 4.13 illustrates the pattern finder, which enables the user to search for target patterns based on one of the eight elementary pattern fields. These are considered the most important features used for pattern identification purposes. MUIP can search patterns based on any subset of these fields. Data entry can be by keywords or full text. The force field can be either the label or any keyword or the full text of a force. The results of the search will again be displayed in the *Search Results* list for further browsing.

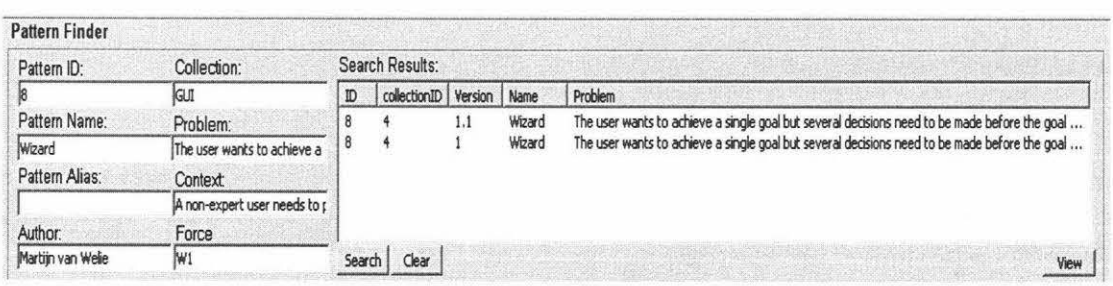


Figure 4.13 Pattern Finder.

Modification (Versioning & customising)

In Figure 4.14, the user can enter a new revision number and create a new version of the pattern in the repository. This way, the user can modify a pattern without changing the original copy. The system will therefore keep track of all versions of a pattern. Another purpose of the versioning feature is to allow the user to customise a pattern for a specific problem domain and store this instantiation in the database for future use. The user can rate the pattern by choosing from the scale "Zero stars" to "Three stars" in the *Confidence* field after reviewing a pattern.

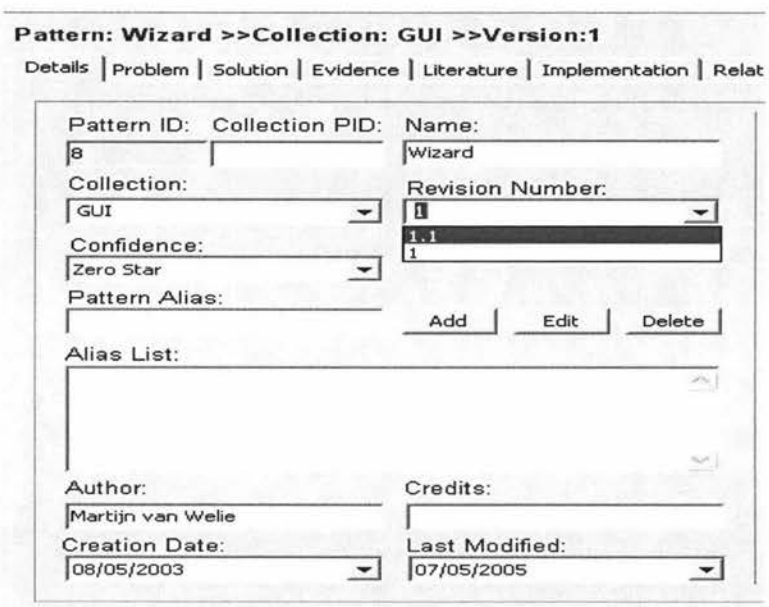


Figure 4.14 Versioning patterns.

A further annotation facility for any change or comments regarding a pattern is the change log. The change log interface, presented in Figure 4.15, enables the user to record notes about a specific modification or to record comments after reviewing a pattern. All log contents are organised into a dated list. The system can provide automatic logging for some important modifications, such as the creation of a new version of a pattern or when a pattern's name is changed.

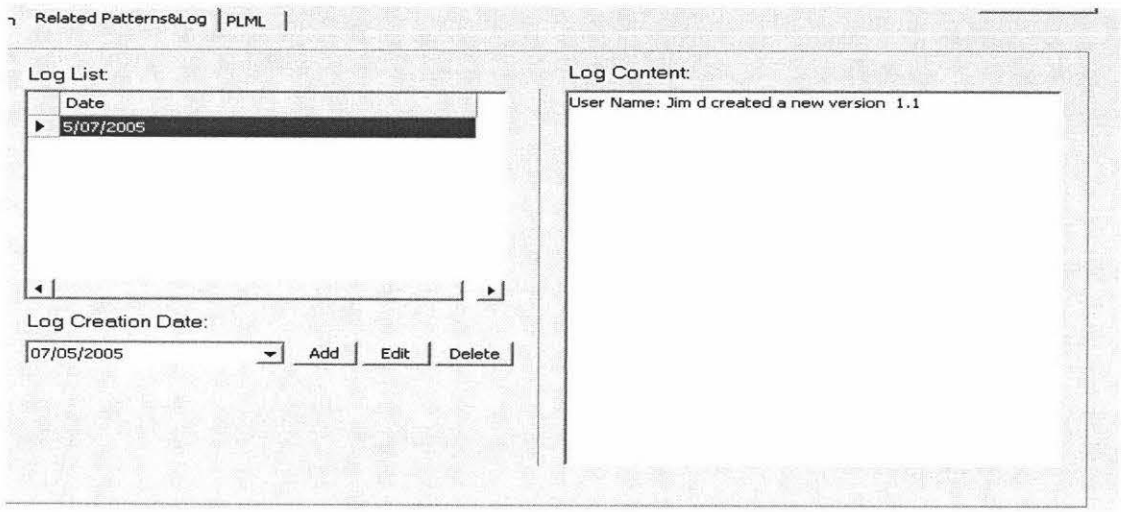


Figure 4.15 Annotations.

Relating patterns

If the user identifies a related pattern by selecting the *Pattern ID, Collection and Version Number* as shown in Figure 4.16, the user can then choose the relationship type and click the *Add* button to create a relationship between the two patterns. The label field allows the user to give a specific name to the relationship. In the Type view, the user can create new types of relationship and store them for later use. This process helps the user explore and accumulate different types of relationships between existing patterns in the repository.

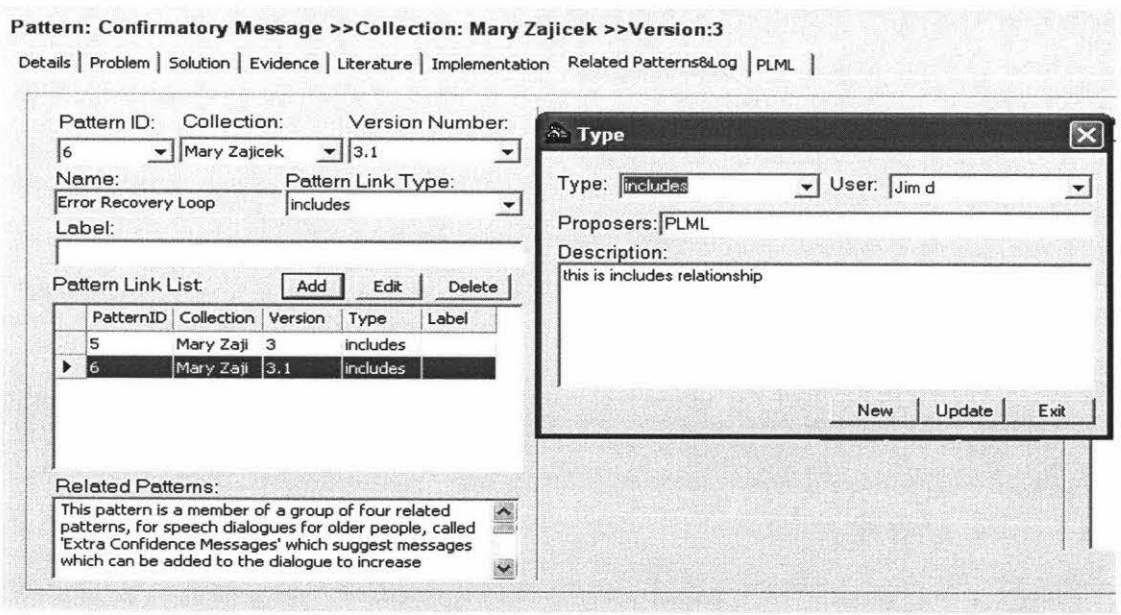


Figure 4.16 Adding relationships and Type view.

Manipulating collections

The collection view allows the user to manipulate UI pattern collections, including the creation of new pattern collections, categorisation of existing patterns into named collections or collecting patterns for personal use (see Figure 4.17). The user can enter a new collection name into the *Name* field and other details as well, and then clicking on the *New* button will save the new collection name into the system. Selecting a collection name from the list, the patterns within the selected collections will be shown in the *Patterns in collection* list. The user can also select a pattern from the *Patterns in system* list and click the *Collect* button so that the selected pattern will be collected into the target collection.

Figure 4.17 Collection View.

Input/Output

The user may want to import patterns from other resources into the system. However, other pattern resources will probably not be written in the PLML v1.2 format. There are two ways to import a pattern into the system. One method is to import a pattern written in PLML v1.2 format automatically. In this case the parser will parse the elementary data and transfer the

data into the appropriate fields in the pattern editor. Besides importing patterns in PLML v1.2 format directly, MUIP allows the user to manually import patterns stored in alternative formats, such as RTF, HTML, TXT, XML or PLML v1.1. Figure 4.18 illustrates this importing process. The user can use the Open file dialog to upload the pattern source file into the tool and the content of the selected file will be displayed in the *Pattern* textbox on the left-hand side. The source file may contain image data. The user selects the information from the *Pattern* textbox without the element heading and selects the appropriate PLML elementary option from the pop-up Context Menu. Then the selected elementary information is pasted into the appropriate field in the pattern editor. In the meantime a PLML file with the selected information data will automatically be generated and displayed in the PLML textbox on the right-hand side of the interface. The selected information will be highlighted in the PLML view so the user can check that the operation was correct. If the mapping operation is wrong, the user can use the *Undo* button to cancel any step of the previous operations.

The patterns from the repository can be exported to files in both RTF format and PLML formats. While a pattern is being viewed in the pattern editor the views are shown: the pattern is displayed in the *Pattern* textbox in RTF format, and in the *PLML* textbox using the PLML format (see Figure 4.18). The user can click the *Export* button under the *Pattern* textbox to export the pattern written in RTF into a file. Two versions of PLML are supported by the system, PLMLv1.1 and PLMLv1.2. The default version is PLMLv1.2 but the user can choose to export the pattern as PLML v1.1 by changing the value of the Combobox, next to *PLML*. Setting the value of *Include Diagram/pictures* with "Yes" will enable PLML to contain the image data so the user can export the pattern with image data in the PLML file.

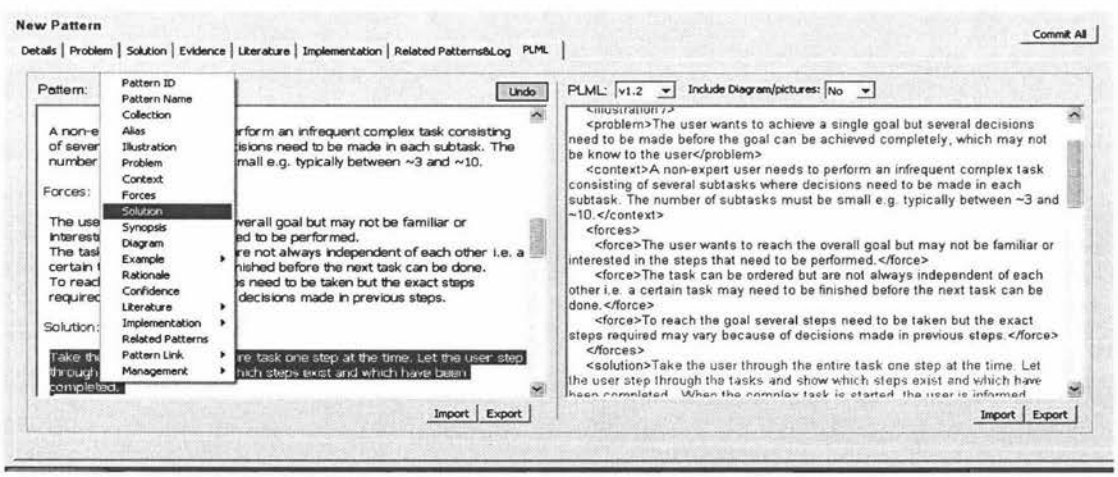


Figure 4.18 Importing pattern in RTF.

4.3 Review of MUIP

Features implemented

Table 4.3 Features implemented (Y indicates "Yes"; N means "No".)

Main Feature / Phase	Domain Features (Identified in Chapter 3)	Requirement specification	Design	Implementation
Authoring	Specific pattern form	Y	Y	Y
	Alternative pattern form	Y	Y	Y
	Input pattern template	Y	Y	Y
	Create new pattern using template	Y	Y	Y
	Support multimedia data	Y	Y	Y
	Include implementation code	Y	Y	Y
Manipulating forces	Create force	Y	Y	Y
	Browse a set of forces	Y	Y	Y
	Identify/reuse forces	Y	Y	Y
Browsing	Display pattern list	Y	Y	Y
	View pattern details	Y	Y	Y
	Different views	Y	Y	Y
Searching	Keyword search	Y	Y	Y
	Full text search	Y	Y	Y
	Search based on different fields	Y	Y	Y
	Search pattern based on any subset of fields	Y	N	N
	Search pattern based on additional data	N	N	N
Modification (Versioning & customizing)	Annotate	Y	Y	Y
	Pattern versioning	Y	Y	Y
	Keep track of customisation of patterns	Y	Y	Y
	Modify patterns	Y	Y	Y
	Rating of patterns	Y	Y	Y
Relating patterns	Create relationship type	Y	Y	Y
	Add relationships	Y	Y	Y
	Naming relationships	Y	Y	Y
	Visualise relationships	Y	N	N
Manipulating collections	Collect a set of patterns for personal use	Y	Y	Y
	Categorise patterns into named collections	Y	Y	Y
	Create new collections	Y	Y	Y
Input/output	Import single pattern	Y	Y	Y
	Import a collection of patterns	Y	N	N
	Export single pattern	Y	Y	Y
	Export a collection of patterns	Y	N	N
	Export pattern map	N	N	N
	Support XML	Y	Y	Y
	Support PLML	Y	Y	Y
	Provide output template	Y	Y	Y
	Support alternative output formats	Y	Y	Y

A software implementation of MUIP requires support for pattern authoring, manipulating forces, browsing patterns, searching patterns, supporting pattern versioning and customisation, relating patterns, manipulating collections and supporting importing and exporting patterns. The detailed domain features have been identified in Chapter 3. The feature specifications of the system and the design issues of these features have been

discussed in the previous sections. Most of these features have been included in the implementation of the system. Table 4.3 shows the detailed features that were identified, designed and implemented in the system.

Features still to be implemented

Due to time constraints some features have not been implemented fully. These features are still important to a pattern management system and they will be considered for implementation in the future work as described in Chapter 6. Table 4.3 also shows which features have not yet been implemented:

- The searching patterns have not been implemented sufficiently. The search function has not been fully enabled to do a search based on any subset of the elementary fields. It has not provided additional filters to support the search function, as MOUDIL has.
- The implementation only supports two types of image types, such as the JPEG and GIF. Other image formats have not yet been supported.
- The visualisation of the relationships between patterns has not been implemented.
- This implementation can only import and export a single pattern each time either in PLML formats or an alternative format.
- The administrative function of a database administrator has not been included in the implementation.

4.4 Summary

This chapter illustrates the MUIP prototype. The conceptualisation of the system was discussed and outlined in Chapter 3. The domain features identified form the foundation for the features specification which guided the implementation of the prototype. A new version of PLML v1.2 was developed to meet the needs identified during the features analysis. Detailed tasks were defined as use cases. The architecture of the system was defined and a conceptual model of the database for storing UI pattern collections in PLMLv1.2 format was designed. The development tools employed to create the prototype were identified. Next the detail of the prototype was presented using screenshots to illustrate each of the features implemented. The last section provides a brief review of MUIP.

A formal evaluation of the prototype is presented next in Chapter 5. This concentrates on evaluating the usefulness of MUIP to identify how to improve the usability of the prototype. The detailed evaluation process and results will be presented.

Chapter 5 Evaluation

This chapter, first, introduces the topic of evaluation, describing evaluation frameworks and related techniques in detail. The following section explains the evaluation methodology adopted to conduct this study. The fully detailed description describes how the DECIDE evaluation framework was applied to guide the evaluation. The next section presents the results of the study and analyses them. Finally, a brief summary is given.

5.1 Background

Evaluation is a critical component of an effective software prototype development process (Jacko & Sears, 2003). Dix *et al.* (2004) pointed out that evaluation cannot be considered a single phase within the design process as it may apply to the whole design life cycle. An effective evaluation will identify interface problems and provide further guidance to improve the current design. In general, three main objectives of evaluation have been recognised (Dix *et al.*, 2004):

- Assessing the extent and accessibility of the system's functionality;
- Evaluating users' experience of the interaction;
- Identifying any specific problems with the system.

Evaluation is applicable in various phases of the software development process. Relating to the different development phases, evaluations can be grouped into two general types: *formative evaluations* and *summative evaluations* (Preece *et al.*, 2002). Formative evaluations are used to assess a system to see if it meets the user's needs during the development process. The assessment can identify the pros and cons of the current design for further constructive improvement. Once the system is developed completely, a full summative evaluation can check how well the product is doing and whether it satisfies the user's requirements and standards.

Conducting an evaluation requires the consideration of all aspects of the evaluation process. A novice or experienced evaluator can benefit from a well-defined and structured guideline for planning an evaluation. Malone *et al.* (1984) summarise a framework for the evaluation of human-computer interfaces:

1. Identify the objective of the evaluation.
2. Identify all aspects to be evaluated, including selection of test participants or subjects.
3. Select specific tasks to be used in the evaluation.
4. Select operational conditions and tasks to be used in the evaluation.
5. Select test methods (techniques).
6. Select test measures, including measures of program user friendliness, program quality, learnability etc.
7. Identify evaluation criteria.
8. Identify requirements for data collection and recording.
9. Identify data validation requirements.
10. Identify requirements for data analysis.

The above framework includes all the main aspects of an evaluation process.

Another brief framework for benchmark evaluation can be used to assess the performance of a prototype (Sutcliffe, 1995):

1. Set usability goals.
2. Develop product or prototype for testing.
3. Design evaluation tasks.
4. Select users.
5. Carry out evaluation.
6. Analyse results using usability metrics.

Compared with the previous framework, this framework gathers the most practical issues into the step "Carry out evaluation", which includes the selection of evaluation techniques. However, neither of them suggest how to deal with ethical issues. Based on these two frameworks, evaluators may not understand how to use the evaluation goals to guide the evaluation process.

In contrast, DECIDE (Preece *et al.*, 2002) is a comprehensive evaluation framework which can aid the evaluator in organising an evaluation procedure efficiently. Preece *et al.* (2002) have emphasised the importance of evaluation goals for planning a well-structured evaluation procedure. Therefore, the evaluation based on the framework should have explicit

goals and specific questions, derived from the goals, to be evaluated. The DECIDE framework is composed of six phases:

1. *Determine* the overall goals that the evaluation addresses.
2. *Explore* the specific questions to be answered.
3. *Choose* the evaluation techniques to answer the questions.
4. *Identify* the practical issues that must be addressed, such as selecting participants.
5. *Decide* how to deal with the ethical issues.
6. *Evaluate,* interpret, and present the data.

This study adopted the DECIDE framework to guide the evaluation process. Focusing on the identified goals and questions, appropriate evaluation techniques can be selected to evaluate the prototype. Practical and related issues may affect the choice of techniques for an evaluation. For instance, a combination of a set of appropriate techniques seems to be the best option but it may be expensive or the equipment may not be available. Therefore, all the constraints and conflicts should be taken into account. The next section summarises the main evaluation techniques and the following sections show how they were applied.

5.1.1 Techniques

A variety of techniques have been explored for supporting the evaluation process. Sears classified the different approaches into three different categories (Jacko & Sears, 2003): inspection-based evaluations, user-based evaluations and model-based evaluations. In inspection-based evaluations, techniques focus on the opinions and judgements of experts, except for the pluralistic walkthrough. User-based evaluation depends on testing the system with the users who will ultimately employ the developed system. Predictive models are used to predict user performance when it is difficult to conduct user tests. In the following section, different techniques (see Table 5.1) selected from the above three categories are discussed and their advantages and disadvantages are outlined (see Table 5.2).

User-based evaluations or techniques are direct and efficient methods used to assess a product by the users (Jacko & Sears, 2003). Users participate directly in user-based evaluations although other kinds of evaluations may include users, such as the pluralistic walkthrough. These evaluations focus on the interaction between the prototype and the users and also collect target users' view points, based on their experience with the prototype. Different users can show different perspectives towards various aspects of the prototype.

They may point out what they want and expect from the product. Many drawbacks of the various aspects of the prototype can be identified by users. This study will adopt a set of user-based techniques to evaluate the prototype. There are brief descriptions of the selected techniques in the following section.

Table 5.1 Techniques

Categories (evaluations)	Techniques
Inspection-based	Cognitive Walkthrough, Cognitive jogthrough, Pluralistic walkthrough, Heuristic evaluation
User-based	Field study, Diary, Interaction logging, Observation, Questionnaire, Unstructured interview, Structured interview, Semi-structured interview, Focus group, Think aloud
Model-based	GOMS model, etc.

Interaction logging

Interaction logging has been used to record the interaction between the software system and users for usability analysis. In general, it records key presses, mouse or other device movements. With database management systems, interaction results can be considered one kind of log. Logs usually contain a time stamp in order to identify how long users spend on a specific task. The collected data can be used together with the data from video and audio logs to aid the evaluators' analysis of user performance and how the users work on the predefined tasks (Preece *et al.*, 2002).

The strength of interaction logging is that a large amount of interaction data can be logged automatically and it can log the users' actions unobtrusively. It is cheap and can be used for long-term studies. However, the ethical issues need to be considered carefully when recording the user's activity unobtrusively. It also requires adopting useful tools to facilitate analysing the collected data efficiently (Preece *et al.*, 2002).

Table 5.2 Summary of techniques

Techniques	Advantages	Disadvantages
Cognitive walkthrough (Jacko & Sears, 2003; Rowley & Rhoades, 1992)	• Put focus on the user. • Focus on known problem areas. • Recognise user's goal .	• May be time-consuming, tedious and laborious. • Make designer and experts as users. • Cover limited problem space.
Cognitive jogthrough (Rowley & Rhoades, 1992)	• Capture all comments and discussion. • Identify key aspect of interface.	• Produce large amount of data. • Require external software or equipment.
Pluralistic walkthrough (Preece *et al.*, 2002)	• Increase the probability of discovering possible problems and resolve them. • Involve different stakeholders.	• Lowest proceeding rate. • Cover limited number of tasks.
Heuristic evaluation (Dix *et al.*, 2004; Jacko & Sears, 2003)	• Use experts. • Focus on known problem spaces. • Flexible and cheap.	• No user participant.
Field study (Bly, 1997; Dix *et al.*, 2004)	• User can work in natural environment.	• Constant interruptions.
Diary (Preece *et al.*, 2002)	• Inexpensive and suitable for long-term study. • No special requirement for the equipment and expertise.	• User may forget to complete diary. • Difficult to ensure the quality of entry data from user.
Interaction logging (Preece *et al.*, 2002)	• Log interaction data automatically and unobtrusively. • Cheap and suitable for long-term study.	• Require specific tools to help analyse collected data.
Observation (Jacko & Sears, 2003)	• Direct way to observe the interaction between system and user. • Easy to identify problems. • Identify the strengths and weakness of the prototype.	• Not easy to infer causality. • Difficult to observe what the observers are interested in. • Participants may change behaviour. • Observers see what they want to see.
Questionnaire (Dix *et al.*, 2004; Stone *et al.*, 2005)	• Collect the view point from participants. • Can distribute to a large number of people and can compare answers from different participants. • Cheap and quick. • Can collect some quantitative data.	• Not always reliable. • Participant may refuse to answer questions. • Difficult to design a well enough questionnaire. • Designers have to predict what the users will concern. • Closed-ended questions give little information regarding why participants choose this answer.
Unstructured interview (Preece *et al.*, 2002)	• Participants feel free to answer questions. • Produce rich amount of data.	• Analysis of data is time-consuming and difficult • Cannot repeat interview process.
Structured interview (Preece *et al.*, 2002)	• Standard study. • Easy to analyse data.	• Difficult to get additional opinions from participants.
Semi-structured interview (Preece *et al.*, 2002)	• Participant can explore more issues.	• Difficult to analyse data.
Focus group (Preece *et al.*, 2002)	• Produce possible design choices and new features. • Low-cost and gets quick results.	• Does not test real interaction between user and prototype. • Record what users think what they want, which may not reflect what they actually use.
Think aloud (Dix *et al.*, 2004; Jacko & Sears, 2003; Stone *et al.*, 2005)	• Close to actual individual usage. • Simple to use. • Get immediate feedback. • Help participants to focus during the evaluation process.	• Unnatural for user. • Collected data may be subjective and selective. • Participants maybe silent without comments. • May slow the participant's thought process and affect the task performance. • Makes participants exhausted.
GOMS model (Preece *et al.*, 2002)	• Allow comparative analyses to be performed for different interfaces or systems.	• Only model some specific types of task. • Only predict expert performance. • Does not model errors.

Observation

Observation involves watching what users do and listening to what they say. Observing the interaction between users and the software system can help evaluators to identify what they do, what is well supported and what support is needed (Dix *et al.* 2004). It is a useful way to gather information regarding the actual use of the system by observing users interacting with it. Generally, users are asked to perform a set of predetermined tasks based on scenarios. The observation can be carried out in a controlled environment, like usability laboratories, or in natural environments, such as the work places of the users.

However, there are several limitations to observation (Jacko & Sears, 2003):

- It is not easy to infer causality while observing any behaviour.
- The observer is unable to control when events occur so that it may take a long time to observe what you are interested in.
- Participants may change their behaviour when they know they are observed.
- Observers usually see what they want to see, which may affect the validity of the observation.

Observation framework

Preece *et al.* (2002) have stated that observation frameworks can aid focus and organise the observation and data collection. There are three observation frameworks discussed by different researchers. The first one is a simple practitioner's framework, which only contains three items (Preece *et al.*, 2002):

- *The person.* Who is using the software system at any particular time?
- *The place.* Where are they using it?
- *The thing.* What are they doing with it?

However, the framework discussed above is too simple as it does not easily help the observer to plan his or her observations. Another more detailed framework was proposed by Goetz and LeCompte (1984), which advises observers to pay more attention to the context of events, the participants and the system:

- *Who* is present and what are their characters? What is their role?
- *What* is happening? What are participants doing and saying and how are they behaving?

- *When* does the action occur? How is it related to other actions?
- *Where* is it happening? Do physical conditions play a role?
- *Why* is it happening?
- *How* is it happening? What rules influence behaviour?

The Goetz and LeCompte framework emphasises the context of events. Another framework with similar items was developed by Colin Robson (1993):

- *Space*: What is the physical place like and how is it laid out?
- *Actors:* What are the names and relevant details of participants involved?
- *Activities:* What are actors doing and why?
- *Objects*: What are the physical objects present?
- *Acts:* What are specific individuals doing?
- *Events:* Is what you observe part of a special event?
- *Goals:* What are the actors trying to accomplish?
- *Feelings:* What is the mood of the group and of individuals?

As outlined above, Robson's framework provides more detail than other two. This framework also pays attention to the mood of the participant group. The Goetz and LeCompte framework focuses on the context of events. Although there is some overlap between the Robson and the Goetz and LeCompte frameworks, the Robson framework seems to be more explicit. It aids observers to organise observations according to these eight aspects.

Questionnaire

Questionnaires are a well-structured technique used to gather demographic information and the users' view points (Preece *et al.*, 2002). A questionnaire can start by asking about the users' background information, which can help categorise them into different sample groups. Both closed-ended and open-ended questions can be asked. If a questionnaire includes too many questions, the questions can be further grouped into sub-level related topics in order to make it easier to read and complete. Questionnaires can be used to reach a large number of participants. They usually get a quick response and take less time to administer and to score. The data collected from questionnaires can be analysed rigorously (Dix *et al.*, 2004). However, developing an effective questionnaire is time-consuming and requires special skills.

Dix *et al.* (2004) summarised a number of styles of questions that can be included in a questionnaire:

- *General*: These questions help establish the background of the user and his/her position within the sample group. These questions may collect data concerning age, gender, occupation, place of residence, experience with computers and other related background information. They may be classified as open-ended, multi-choice or scalar questions.
- *Open-ended:* These questions ask for the users' subjective viewpoint regarding a specific issue. They are useful to gather subjective opinion from users but are difficult to analyse. These responses may contain suggestions for the further improvement of the system.
- *Scalar:* Users are asked to judge a specific statement on a numeric scale. The scale corresponds to a measure of the user's agreement or disagreement with the statement. The granularity of scale is various: a coarse scale (from 1 to 3), the Likert Scale (from 1 to 5), and so on.
- *Multi-choice:* Users are offered a set of explicit responses and may be asked to select one or more of the answers, if applicable.
- *Ranked:* These questions require users to place an order on items in a list.

5.1.2 Triangulation

Triangulation is a general term that describes how different perspectives can be used to improve the understanding of a problem and situation. Therefore, triangulation can combine different techniques to get different perspectives or to examine data in different ways (Preece *et al.*, 2002). Triangulation can be used to check the validity of the evaluation study and to identify from different perspectives whether the findings are true or not.

5.2 Methodology

The purpose of this evaluation was to assess whether the prototype can support researchers or UI designers in managing pattern collections efficiently and to identify the usability problems for further improvement. Therefore, the evaluation was a formative evaluation. This evaluation plan was based on the DECIDE framework, covering all issues involved in the evaluation process.

5.2.1 Determining Goals

In this research a prototype for manipulating pattern collections was developed by the researcher. The prototype, MUIP, contained features like exchanging pattern data, versioning patterns, manipulating pattern collections, manipulating forces and the support of related patterns. The main goals of this study were:

- G1: To check that the users can complete appropriate tasks regarding managing UI pattern collections.
- G2: To find out how to improve the usability of the prototype.

5.2.2 Exploring Questions for the Goals

To satisfy the goals, a main question for the evaluation was identified: "Do users use the tool efficiently to manage UI pattern collections?" The main question can be broken down into the following key sub-questions for evaluation:

1. Can users exchange pattern data efficiently?
2. Does the prototype support versioning patterns?
3. Can users manipulate pattern collections by using the prototype?
4. Is it easy to manipulate forces?
5. Does MUIP support creating relationships between patterns?
6. What are the strengths and weakness of the prototype?
7. What do users think about the prototype?

Questions 1 to 5 are related to G1 and they will be answered to examine this goal. The answers for questions 6 and 7 will provide information on how to improve the usability of the prototype, as described in G2.

5.2.3 Choosing Evaluation Techniques

As discussed in the previous section, there is a list of techniques, which could be used in the study. Inspection-based techniques usually require experts to carry out the evaluation. However, it is difficult to hire professionals to evaluate the prototype due to financial constraints. Three user-based techniques were selected for this evaluation: observation, questionnaire, and interaction logging. They were combined to identify various perspectives

of the prototype and the different types of data from various techniques show different points of view. Therefore, using triangulation can reveal a broad picture of the prototype.

Observation is a direct method to see how well the prototype meets the user's needs and how to improve the usability of the prototype. Observations help evaluators to identify the usability problems efficiently. Observations can be held in a well-equipped, controlled laboratory or natural working environment. However, the researcher's department did not have a usability laboratory available at the time of the study. On the other hand, the prototype is proposed for use in a natural environment to help researchers or user interface designers to manage pattern collections. Therefore, it is necessary to observe how the prototype can help participants complete their tasks in a natural working environment.

Questionnaires are an efficient technique to gather information on the participants' opinions and experiences. The evaluator can get a quick response from a small group of participants. The quantitative data from the predetermined questions can be calculated and analysed easily to draw conclusions about the participants' viewpoints on the prototype. Open-ended questions can gather helpful advice and suggestions regarding the features and usability of the prototype from participants.

MUIP is a database management system so it can record the result of the interaction between the system and the participants. If participants choose to save data into a backup repository, most interaction data will be stored. The system also provides automatic logs for some specific actions. Thus, the system itself can act as a handy logging tool to collect data for the interaction logging technique. It is an inexpensive and easy way to gather interaction data. The data can be used to help answer the questions 1 to 5 in the previous section.

5.2.4 Identifying the Practical Issues

There are various practical issues to be considered and identified before starting the evaluation study. Those practical issues include the selection of participants, data collection (the development of the observation framework/protocol), and evaluation documents.

Participants

Five computer science staff members familiar with patterns and UI related patterns joined in the evaluation of MUIP. They were experts in either HCI or software engineering (SE).

Table 5.3 Summary of participants

Participant No.	Research interest	Familiar level with concept of patterns	Familiar level with concept of UI related patterns
1	E-learning	Familiar	Familiar
2	Programming Methodology	Very familiar	Familiar
3	Knowledge bases, OO Design, Web application Design	Very familiar	Very Familiar
4	HCI	Very familiar	Very Familiar
5	HCI, SE	Very Familiar	Very Familiar

Data collection

From the evaluation goals and questions, the author identified the main tasks a potential user may undertake while manipulating pattern collections with this prototype. Based on the previous sub-questions, the author designed five scenarios for participants to perform. All participants would be observed performing the first three scenarios.

Preece *et al.* (2002) have noted that observation frameworks can aid focus and organise the observation and data collection. There are three observation frameworks discussed by different researchers. The author adopted Robson's framework, which includes the following aspects: space, actors, activities, objects, acts, events, goals and feelings. Robson's framework is quite explicit and it is an easy aid to organising the observation. This evaluation focuses on the usefulness of the prototype and observes the emotional feelings. Such an observation requires sophisticated methods and equipment. Thus, the evaluation process did not assess the feelings of participants. The author organised the observation as follows:

- *Actors*: As described in the previous paragraph, five researchers were selected to take part in the evaluation, each of whom had a solid knowledge of UI patterns and the HCI area. The author acted as the observer during the evaluation process.
- *Space:* Participants performed their tasks in their own natural working environments. Such a working environment would let participants use the system naturally and comfortably.

- *Activities:* In each evaluation session, there were a participant and an observer. The participant explored how to use the system for ten minutes at first. This exploration process helped the participant get used to the interface of the prototype. After the exploration, the participant performed five main tasks based on the scenarios provided. During this period, the participant informed the observer of their thoughts about the prototype at any time. From the observation of the interaction, the participant identified the strengths and weaknesses of the prototype. The observer recorded related data for the first three scenarios. The time to complete each scenario was recorded. The observer took notes in observation form regarding the system's performance (such as errors, slips and requests for help) and the participant's thoughts.
- *Objects*: The objects included in this evaluation were white paper, pens, predefined document materials, and a computer installed with the relevant software. The studies were carried out using the author's personal laptop computer. There are a few reasons why the author's computer was selected rather than the participants' computers. First, the installation required the use of other software packages to execute the prototype. However, some participants were not keen to install additional external software packages onto their computers. Second, the computers of participants may vary. Some had high quality computers and others may have had a computer with a lower level of performance. To ensure that participants explored the prototype in a similar environment, it was better to use the same computer. Third, using the author's computer made it is easier to collect the interaction logging data.
- *Acts:* Under observation, the participant performed the tasks specified in the first three scenarios. The other two scenarios were accomplished by the participant individually, without observation. Once the participant finished the testing, s/he was requested to answer the questionnaire, which included closed and open-ended questions. The evaluator picked up the questionnaire from the participant when it was ready.
- *Events:* The observation focused on: how long the participant spent on each observed tasks; what the participant thought; what the participant looked at; what the participant found confusing; how the system performed: number of errors, number of slips, and tasks completed.
- *Goals:* The actors complete their activities as discussed above.

Participants performed the tasks in their working environments. The author acted as the observer to observe what participants did and noted comments from the participants. To facilitate the process of taking notes, a template for observation of frequent activities was developed. The template included the start time, completed time, total number of errors and total number of slips for the scenario. It was used to record data for the three scenarios. There was a table for recording the main interaction data for the scenario. The table was composed of three columns. The column "Main tasks" presented a sequence of main tasks for each scenario. The column "Action" recorded the data related to help, slips and errors for appropriate tasks. The column "Notes" was available to take notes regarding the help, slips and errors and the related context of the interaction.

All participants were requested to answer a predefined questionnaire at the end of the evaluation session. The Likert Scale (from 1 to 5) was used in this questionnaire where 1 represented strongly agree and 5 represented strongly disagree. Most closed questions in the questionnaire adopted the Likert Scale for responses, except certain questions (item 10, 12, 13, 22). These four items provided two options (Yes/No) for the answer. The items from 1 to 9 in the questionnaire were adopted and modified from the Software Usability Measurement Inventory (Kirakowski, 1994), which concerns issues such as efficiency, affect, helpfulness, control and learnability. Questions from 10 to 26 were related to the specific functionality of the tool. The last part of the questionnaire contained four open-ended questions, which focused on suggestions regarding the features of the tool, the pros and cons of the tool and further comments regarding the prototype (see Table 5.6).

The interaction data can be recorded by use of the backup database of MUIP, such as new patterns, new versions, and change logs, etc. For example, if the participant created and saved a new pattern into the system, the tool recorded the appropriate content of the pattern. Automatic logging supported some specific actions such as modifying some critical fields of a pattern and creating new versions. The records of the database reflected the interaction between participants and the tool from a different angle. For each evaluation session, a copy of the backup database was saved as an evidence source and subsequently analysed.

Evidence for answering the evaluation questions

The analysis of data aimed to provide the answers to the evaluation questions. To prepare the analysis, all collected data was reviewed. The evidence in this evaluation came from three

kinds of data: observation protocol, questionnaire and the database record. Table 5.4 illustrates the database record. The database recorded changes made by participants.

Table 5.4 Result table of database

Change / Participant #	D1	D2	D3	D4	D5
	Number of new patterns	Number of new collections	Number of new copies of patterns in collections	Number of new versions of patterns	Number of new relationships between patterns

Table 5.5 shows which sources of evidence were used to answer the sub-aspects of the main evaluation questions.

Table 5.5 Sub-aspects of evaluation question with main source of data

Key aspects	Questions in questionnaire		Observation protocol	Database
	Closed-ended	Open-ended		
Can users exchange pattern data efficiently?	10,11,12,13	27	Scenario 3	D1
Does the prototype support versioning patterns?	15	27		D4
Can users manipulate pattern collections by using the prototype?	14,16,17,18, 19,20,26	27		D2,D3
Is it easy to manipulate forces?	21,22	27	Scenario 1	D1
Does it support relating patterns?	23,24,25	27	Scenario 2	D5
What are the strengths and weakness of the prototype?	1,2,3,4,5,7,8,9	27,28,29,30		
What do users think about the prototype?	1,2,3,4,5,6,7,8,9	27,28,29,30		

Documents for evaluation

From the above discussion, six documents were required to be prepared to conduct the evaluation: Handy Hints, Information Sheet, Informed Consent Form, Observation Form, Task Scenarios, and Questionnaire. The details of each document can be found in Appendices E -J.

5.2.5 Deciding Ethical Issues

Informed consent (See Appendix J) was obtained from the participants before the evaluation procedure commenced. The goals and what they should expect were explained clearly in

advance. All participants were told that it was the system that was being evaluated and not their performance.

5.3 Pilot Studies and Evaluation

To make sure the evaluation procedure was working properly, five pilot studies were completed before launching into the formal evaluation. These pilot studies helped improve the initial design of the evaluation. It helped ensure the materials used in the study were clear and that the experimental procedure was working properly. A few small problems within the tool were identified and fixed before the main study.

5.3.1 Changes to Documents

Handy Hints

Based on the pilot studies, some participants had a problem finding the target tab in the interface. Therefore, specific names of data fields were added to a tab name. The help for importing a pattern manually was found to be unclear. The work flow for importing a pattern manually was revised so the process was clearer.

Information Sheet

The previous version of this document did not contain the statement of the evaluation's purpose. One participant advised adding this statement into this sheet. Thus, the document contained the goal of this study in the final version.

Task Scenarios

The instructions for the ten minute exploration were found to be unclear. Some participants found it difficult to follow the instruction that enabled them to logon to the system. This instruction was made clearer by adding additional information.

Some participants had difficulties understanding the procedure for Scenario 3. Additional explanation information related to the specific user interface was added into the brief description section of the scenarios. The basic flow of the scenario was revised so it could be more easily understood.

Table 5.6 Final questionnaire for the evaluation

No.	Item	Strongly agree (1)	Agree (2)	Ok (3)	Disagree (4)	Strongly disagree (5)
1	Working with MUIP is easy.					
2	It takes too much time to learn how to use it.					
3	It provides enough guidance to perform tasks.					
4	The help documentation makes me confused.					
5	It is straightforward to get the task done.					
6	Error prevention messages are adequate.					
7	The response from the system is too slow.					
8	The interface of MUIP is logical and attractive.					
9	I would like to use MUIP to manage my own UI pattern collections.					
10	MUIP can import patterns written in RTF format.	Yes		No		
11	It is complex to convert other pattern forms into PLML.					
12	MUIP can export patterns written in PLML.	Yes		No		
13	MUIP can export patterns written in alternative formats.	Yes		No		
14	Editing the content of a pattern is convenient.					
15	It is useful to have multiple versions for a pattern.					
16	MUIP supports sufficient search functions to find target patterns from the system.					
17	A pattern in the system can be found in different ways.					
18	It takes many steps to create a new collection.					
19	Navigating patterns within a collection is difficult.					
20	The way of manipulating collections is confusing.					
21	It is handy to browse existing forces.					
22	MUIP allows a user to reuse a force from the repository.	Yes		No		
23	It is obvious that users can create relationships between any two patterns.					
24	Naming new types of relationship between patterns is difficult.					
25	MUIP provides enough functions to assist me exploring relationship between patterns.					
26	MUIP provides enough functions to manipulate pattern collections.					
	Open-ended questions					
27	What features do you suggest adding to MUIP to further support managing pattern collections?					
28	Would you like to outline the advantages of the system?					
29	Would you like to outline disadvantages of the system?					
30	Would you like to write further comments regarding this system?					

Questionnaire

In the closed-ended questions, the original questions did not contain the Likert Scale on every page. As a result, participants needed to look back to the first page to find out the meaning of the Likert Scale options. Thus the Likert Scale 1 to 5, with its meaning, was added into those pages with closed-ended questions. Four closed-ended questions were found to be using the wrong scale. They were changed to two options: "Yes" or "No" (see Table 5.6).

Two closed-ended questions could not be answered by participants because they had not done the tasks based on the scenarios. These two closed-ended questions were removed from the questionnaire.

In the old questionnaire, there were seven open-ended questions. Four of them were found to be too specific with the result that most of participants did not know how to answer them. Those specific questions were deleted from the questionnaires. One of the remaining open-ended questions was separated into two questions.

Change to the prototype

One participant found that there were two spelling mistakes in the interface regarding "Collection". These spelling mistakes were corrected. In the "Collection" window, the name of a pattern collection could be updated with null. This logical mistake was changed by modifying the code.

Some participants wanted to clear the search fields in the Pattern Finder section more easily. Users had to clear all search fields manually in the old prototype. It was very inconvenient when the users wanted to do the search based on new entry data. Therefore, a "Clear" button was added next to the "Search" button for clearing all search fields.

5.3.2 Evaluate, Interpret, and Present the Data

The evaluation process was conducted with five different participants independently. The result indicated that the prototype, MUIP, provided sufficient functionality to support pattern collection management. Most of the participants agreed that the functionality of managing pattern collections was well supported. However, they also indicated that the interface of the

prototype required improvement. Many recommendations were related to the interface issues of the prototype.

Table 5.7 Results of observation protocol

Participant No.	Scenario No.	No. of Complete tasks/Total No. of tasks	Help requests	Slips	Errors	Time (minutes)
1	1	6/6	2	1		10
2		6/6				4
3		6/6				3
4		6/6	2	1		3
5		5/6	2	1		7
Mean						**5.4**
1	2	4/4	1	1		6
2		4/4		1		5
3		4/4				3
4		4/4	1			10
5		4/4	4			13
Mean						**7.4**
1	3	4/4	1	2		10
2		4/4				8
3		4/4				6
4		4/4	2	1		11
5		4/4	4	1		12
Mean						**9.4**

Observation

Table 5.7 shows that most of the tasks in the three scenarios were completed by the different participants. Only one participant could not finish one task from scenario 1. The person required more help than the others because he spent less time exploring the system before performing the tasks. Only two participants did not refer to help in completing all the tasks, the others needed to read the predefined Handy Hints to aid completing the scenarios. All tasks were completed without errors but with some slips, which were mainly related to inappropriate actions. Participants spent more time in doing scenario 3 than the other two because it was the most complex scenario. The mean time for completing scenario 3 was 9.4 minutes and other two were 5.4 minutes and 7.4 minutes. Some people needed more time accessing help in order to complete the scenarios.

Some common problems were identified in the analysis of the observation protocol for Scenario 1. In searching target patterns, most participants could not find the result because they typed in the wrong search data in the first place. Two persons searched a pattern based on a pattern ID that did not exist in the database. Two participants wanted to use the "Enter" key in the keyboard to trigger the search event, but the system did not support it. They also

found that the search data entered were hidden due to the multiline property of the search area once they clicked the "Enter" key. Two participants complained that it required too much data entry when searching for patterns. One of them claimed that it provided a comprehensive function to browse different patterns from the repository.

All participants had shown their interest in creating relationships between patterns while performing tasks from Scenario 2. They all pointed out that it was difficult to identify relationships among patterns. All of them finally compared the contents of two patterns to explore relationships between them. One participant noted that the order of the relationships explained in the user interface between two patterns was not clear. A participant suggested MUIP should support creating multiple relationships between two specific patterns. The participant advised to separate the "Related Patterns and Log" into two different tabs.

Converting patterns from other formats into the PLML format proved to be efficient (Scenario 3). Two participants showed their appreciation of this function. Although a few participants were hesitant in finding some sub-level hidden options, all of them could use the context menu to convert pattern contents into the PLML format quickly. A few participants used the "Undo" function to correct a wrong operation. However, two users selected all the forces of a pattern together and considered them as a single unit. The participants were unfamiliar with the PLML format, so they spent more time finding sub-level options within the context menu.

Table 5.8 Analysis of data of scenario 3

Participant No.	Scenario No.	Help requests	Complete Successfully	Completion Rate	Slips	Error	Time
1	3	1	Yes	80%	2		10 minutes
2	3		Yes	100%			8 minutes
3	3		Yes	30%			6 minutes
4	3	2	Yes	100%	1		11 minutes
5	3	3	Yes	90%	1		12 minutes
Total		6			4		
Mean							9.4 minutes

(The completion rate means the percentage successfully converted into PLML from the pattern)

From the analysis of Table 5.8, the mean time to complete Scenario 3 was 9.4 minutes. One participant spent 6 minutes to complete all tasks of Scenario 3 and another participant spent 8 minutes in completing those tasks. Three participants had trouble in completing the tasks and requested help. Those three participants spent more time in finishing the tasks, and all of them took more than 10 minutes. A high completion rate for this scenario required more time. The participant who spent 6 minutes on this task only completed 30% of the whole task. Participants, in total, made 6 requests for helps and made 4 slips. The other two participants spent less than 8 minutes on this scenario without any helps and mistakes. Although some participants made some slips, all of them completed the tasks without any error.

Table 5.9 Observation notes for scenario 3

Scenario 3		**Participant 1**
		Look for "Where is PLML tab"
	Help	"Why does the tool have another import button, what is it usd for"
		Put the "Problem" data into the "Context", then do the Undo function
		Select all forces together and choose "Forces" option. Do Undo and select them separately. Comment: "Forces" option should be "Force".
		Enter pattern name and revision number, then select collection and click "Commit All" to save the data
Scenario 3		**Participant 2**
		Select pieces of the content of the pattern by order.
		Select forces separately.
		Hesitated to find the "Known Uses".
		Wondered that what the "Collection" name is. Then select "GUI" and save it.
Scenario 3		**Participant 3**
		Select pieces of contents the pattern by the order.
		Comment: "It is useful to have this function"
Scenario 3		**Participant 4**
	Help	"Where is the Author option?"
	Help	"What's PLML?"
		Selected all of forces the first time. Do undo and select force one by one.
		Hesitated to find the "Known Uses".
Scenario 3		**Participant 5**
	Help	"Where is the Author option?"
		Selected data with heading put "Context" into "Solution" and did Undo.
	Help	"Where is the example diagram?"
	Help	"Where is the version number I should enter?" when the user saved the data.
		Comment: "Need some feedback when the pattern is saved."
		Comment: " 'Commit' should be 'Save'."

The analysis of data from Table 5.9 indicates that two participants had trouble in finding the hidden option for the first time. Another common problem found was that participants put data into the wrong option. For example, one participant put the context into the solution field. Some participants selected all forces together and considered them as a whole.

However, most of the participants knew the method to correct wrong operations by using the "Undo" function. One participant asked for help regarding how to deal with media data, such as a picture or diagram. While saving the new pattern into the database, one participant did not know where to enter the revision number for the pattern for the first time. All participants knew how to select a collection name for the new pattern and eventually save the new pattern into the system successfully.

Questionnaire

Some trends were identified from the answers to closed-ended questions in the questionnaire. The medians for questions 1, 2, 3, 7, and 9 were three, representing an acceptable level of satisfaction regarding the general aspects of the interface (see Table 5.10). The result of question 4 showed that the help documents for the prototype were clear. Most participants agreed that the prototype provided enough error prevention messages, based on the result of question 6. The questions from 10 to 26 in the questionnaire were related to the functionality of the tool. It can be seen from Table 5.10 that 13 questions had positive responses. This indicated that the tool can support these functionalities. Four questions had the median value of three.

Participants agreed the tool provided enough functions to manipulate the pattern collections, as seen in the response to question 26. The result of question 10 in the questionnaire is "Yes", which indicates that the participants agreed with the statement that MUIP could import patterns written in RTF format. Most participants agreed with the proposition that it was complex to convert other pattern forms into PLML, as indicated by the result of question 11. This point outlined that it required a feature to aid the support of converting other pattern forms into PLML. It also indicated that a more convenient method should be applied to facilitate the conversion process. Based on the results of question 12 and 13, the participants agreed that MUIP could export patterns written in PLML format or in alternative formats. From the result for question 15, participants agreed that the prototype was useful in supporting multiple versions of a pattern within the tool. The medians for questions 18 and 19 were three, which showed the method to create new pattern collection was not complex and the function for navigating patterns within collections was acceptable. The result for question 21 of the questionnaire was three for the value of median. Two participants strongly agreed that it was handy to browse existing forces. Others considered this statement as "OK". According to the answers for question 22, all participants agreed with the statement that the system allowed a user to reuse a force from the repository. The result of question 23 in the

questionnaire was two for the median, indicating that the system could create relationships between any two patterns. Based on the result of question 24, participants disagreed that naming new types of relationship between patterns was difficult. The median for question 25 was two, which revealed that participants agreed the prototype provided enough functions to assist exploring relationships between patterns. The results of question 5 and 8 were four, which indicated that the interface of the prototype might be improved. The answer to question 20 showed that the participants were confused about how to manipulate collections. This was the only negative statement regarding functionality.

The results of the closed-ended questions were related to the open-ended questions. The answers to the open-ended questions reflected the trends found in the closed-ended questions. The detailed responses of open-ended questions are described in the paragraph below.

Table 5.10 Results of closed-ended questions.

Q / P#	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26
1	2	3	1	5	2	1	3	2	2	Y	4	Y	Y	1	1	2	1	3	4	4	1	Y	2	4	2	1
2	3	2	3	4	3	2	2	4	3	Y	4	Y	Y	2	2	2	2	5	3	2	1	Y	2	4	2	2
3	3	3	3	4	4	4	5	4	3	Y	3	Y	Y	3	1	4	1	3	5	4	3	Y	1	2	2	2
4	5	4	5	2	4	2	4	4	4	Y	1	Y	Y	3	2	4	2	4	1	2	3	Y	2	2	4	4
5	3	3	4		4	4	2	3		Y	1	Y	Y	2	2	2	2	1	2	2	3	Y	4	5	1	2
Median	3	3	3	4	4	2	3	4	3	Y	3	Y	Y	2	2	2	2	3	3	2	3	Y	2	4	2	2
	m	m	m	y	x	y	m	x	m	y	m	y	y	y	y	y	y	m	m	x	m	y	y	y	y	y

Q: Question P#: Participant No.
x means the result of the question is below the medium value, therefore tending to be a poorer rating.
y means result of the question is above the medium value, tending to be a better rating.
m means the result of the question is the medium value.

Table 5.11 presents the detailed responses from the open-ended questions. These included suggestions for further features, the advantages and the disadvantages of the current design and other comments regarding the system. The participants identified that MUIP was useful to manipulate UI pattern collections. UI patterns in the system were easy to search and browse. The system was able to import or export UI patterns in various formats. However, there were a number of criticisms. Most issues raised by the participants related to the current interface design, while there were fewer recommendations for additional features in managing pattern collections. Two of the participants disliked the professional database phrase "Commit/Commit All" and they advised the use of the normal phrase "Save/Save All". Only one participant noted that the system sometimes responded slowly. Two participants pointed out that the interface was not always intuitive due to a lack of information on the screen and some functionality was hidden. For example, importing a

pattern from RTF format is a little tedious due to hidden options. One participant suggested improving the log function within the prototype so the logs contained more detailed data which could be categorised into different types of logs. One participant recommended using a visual navigator to make browsing patterns from the collections easier and further facilitate the understanding of relationships between patterns. Due to a lack of a buffering tier between the prototype and the backup database, the system could not save the edited pattern content automatically if a user forgot to save it.

Table 5.11 Results of open-ended questions

Question No.	Recommendations
27: further features adding to MUIP	• Improving loggings with more details. Each log may contain when it happens, what happens and who is involved. Most interaction actions should be logged, such as creating patterns or pattern collections, modifications and deleting patterns. The log section may be separated into a different tab. • Providing a buffering tier between prototype and backup database or at least providing a warning message for missing storage of new changes to a pattern. • Manipulating style of interface would make it easier to use, including the improvement of labelling of buttons/options and other UI issues. • Supporting visual navigator to browse patterns and visualise the relationship between patterns. List form may aid navigating patterns from the repository.
28: advantages of the system	• Patterns in the repository can be easily searched and browsed. • Information about UI patterns is sufficient. • It has the ability to import and export patterns in different formats. • The concept of manipulating pattern collections is useful. • Consistency is empowered by the database foreign keys constraints. • Minimum number of screens, but this requires close attention to the grouping of work areas.
29: disadvantages of the system	• It is difficult to understand the meaning of "Commit" or "Commit All". • Sometimes the system responds slowly. • Interface is not always intuitive. Importing a pattern from RTF format is a bit tedious with hidden options. Sometimes it is difficult to understand the way it worked from the interface due to a lack of information on the screen. The tool could be used to achieve the goals, but it was not apparent how to use them to achieve the goals, sometimes. • Users had to focus on the screen too much at times because it was not always clear which area to use.
30: further comments regarding the system.	• Some functionality is a bit hidden. Overall system is useful, but it does take some time to get used to. • In the Register interface, the "Commit" should be in the left side of the window. • In the pattern link section, Pattern ID, Collection and Version Number should have default values. This should allow multiple relationships between patterns. A label should be used to indicate that the relationships add to the existing pattern. • The system may support the XML pretty printing style for PLML. • It was suggested using Visual Studio.Net or Microsoft standard ICON library. • It requires too much entry and correct spelling from the user. Use menus or drop-downs to avoid this. • Screens are clear, but some work is required on grouping or the arrangement of interface and the labelling of options. • Clicking the right key of the mouse to pop up a new pattern content window is unusual. It should make the Pattern ID, Collection and Version Number clear in the title of the pop up pattern content window. • It was advised to use New Zealand date format "dd/mm/yy" and add the title to all pop up messages.

Interaction logging

The interaction results between the system and participants were recorded on the backup database. All participants created one new UI pattern in the system (see Table 5.12). Each participant created at least one new collection and two of them created two new collections. A few patterns (ranging from 2 to 7) were copied into new collections by the different participants. A new version of a specific pattern was generated by each participant. All participants had built a relationship between two patterns, but only three of them saved the relationship into the database.

Table 5.12 Results of database logging

Change / Participant #	D1	D2	D3	D4	D5
	Number of new pattern	Number of new collection	Number of new copies of patterns in collections	Number of new version of patterns	Number of new relationships between patterns
1	1	2	3	1	1
2	1	1	4	1	1
3	1	1	2	1	
4	1	2	7	1	1
5	1	1	2	1	

5.3.3 Evaluation Summary

From the previous section, it is clear that a major strength of the prototype is the useful way it manipulated collections. The results of the observation (see Tables 5.7 and 5.8) showed that the participants knew how to use MUIP to manipulate pattern collections. They could browse patterns, import them, relate patterns and manipulate collections. The responses to closed-ended questions too, indicated MUIP provides sufficient functionality to manipulate pattern collections. The answers to open-ended questions also indicated that the concept of manipulating pattern collections was useful. The data from interaction loggings further revealed that the participants were able to complete tasks such as creating new patterns and collections.

On the other hand, the analysis showed a need to improve some aspects, like interface issues, to further facilitate the process of managing pattern collections. Through the observation, participants sometimes needed help (see Table 5.7) to complete associated tasks, due to a lack of enough interaction information on the screen. The results from both the observation

and the questionnaire indicated the interface of the prototype was not always intuitive (see Table 5.9 and 5.11). Some functionality was hidden so that it was sometimes difficult to understand the way it worked from the interface. Furthermore, some functionality was not supported well, such as the visualisation of pattern relationships mentioned in the responses to question 27. The results of the observations and the questionnaire showed that MUIP only supported one relationship between two patterns so it did not enable the creation of multiple relationships between them.

5.3.4 Changes to MUIP Caused by the Evaluation

Whilst some problems were identified during the evaluation session, the feedback from the participants provided suggestions and recommendations for further improvement (see Table 5.11). A selected set of them (see Figure 5.1) has been applied to improving the prototype based on their importance for improving the understandability of MUIP and aiding the user to manage the UI pattern collections efficiently. From the results of the observations and the questionnaires, some interface issues were found to be more critical than others because they affected how efficiently the participants used MUIP to manipulate pattern collections. The search function is one of the key features of MUIP, which is strongly related to other features. A good search function will make other activities regarding manipulating pattern collections much easier. Relating patterns is also important as it allows associated patterns to be linked within the system. Two participants had trouble using this function. The criteria for selecting and making these changes was based on the degree to which the change improved the usability of the prototype. The following paragraphs describe these changes.

Search fields

Searching target patterns was a very useful function in this system. The initial design of MUIP required users to type search data for any search field manually. However, from the observation data, the users found that they might not know what they can search for within the repository. All patterns inside the system have a unique pattern ID and belong to specific pattern collections. In open-ended question 30, participants advised the use of drop-down lists to display pattern IDs and collection names for selection to save time when entering and avoid spelling mistakes. These two text fields have been changed into drop down lists, which contain the existing data from the repository.

All search fields have been changed to use a single line property. In the initial version, the search data entered would be hidden due to the multiline property of the search fields if the participants clicked the "Enter" key at the keyboard. Due to the changes, clicking the "Enter" key on the keyboard triggers the searching event within the Pattern Finder section.

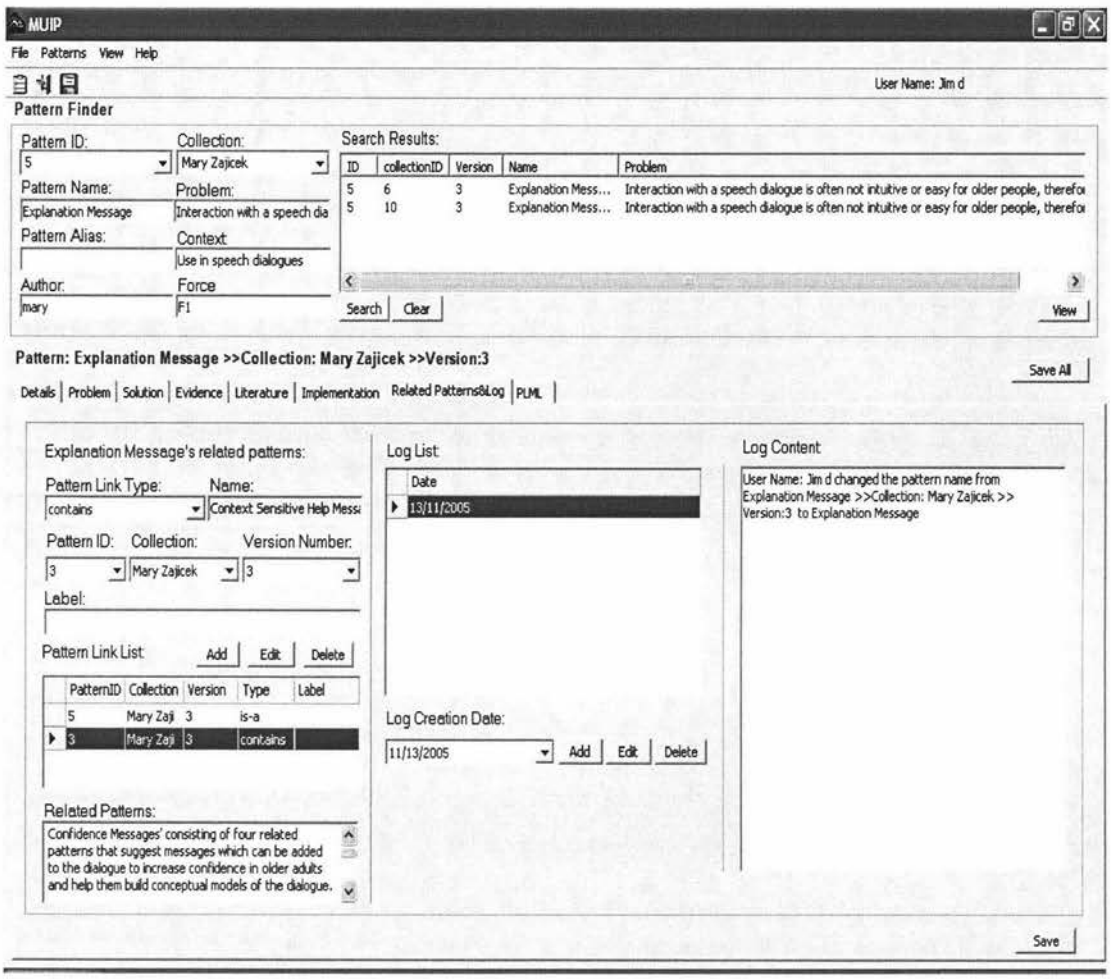


Figure 5.1 The interface of MUIP with the changes made.

Pattern Links

One of the key features of the system was to facilitate the exploration of relationships between patterns. From the result of the observations, two participants found that they did not know how to identify a related pattern in the pattern link section. The default values may show users how to identify a target pattern by selecting the appropriate values of Pattern ID, Collection and Version Number. Default values for those three fields have been set. Additionally, one participant was confused about the order of the relationship. A label, with text, has been used to explain the order of the relationship as advised in a comment on the open-ended questions.

The initial idea was only to support one kind of relationship between two specific patterns. The observational data showed that one participant wanted to create more than one relationship between some patterns. A related suggestion was also made in the answers to the open-ended questionnaire. Therefore, MUIP has been modified to support multiple relationships between any two patterns.

Displaying the pattern content windows

The initial prototype displayed a pattern content window by clicking the right key of the mouse in the Search Results section. The observation data recorded that one participant thought that this action was usually used to show the context menu. Therefore, the action was changed to double clicking the left key of the mouse to pop-up the pattern content window based on the advice from a response to question 30.

If many pattern content windows were displayed at the same time, one participant felt it was difficult to recognise them because the windows did not show the pattern ID in the title of the windows. This recommendation was obtained during the observation as well as from the answers to the questionnaire data. Thus, the pattern ID has been added into the tile of the windows accordingly.

"Commit/Commit all" buttons

Commit is a specific word for database management systems that general users will have difficulty in understanding. Observation found that one participant felt confused about its meaning and two of the participants showed that they were unfamiliar with buttons named "Commit/Commit all". Most common tools, such as Microsoft Word, use the specific word "Save". According to the suggestions in the questionnaire responses, all "Commit" or "Commit all" buttons have been renamed as "Save" or "Save all" buttons.

In the Register window, the "Commit" button was found on the right hand side of the "Cancel" button. The evidence from both the observations and the questionnaire stated that the "Cancel" button usually appeared on the right hand side of all other buttons. The positions of the two buttons have been swapped as well as changing the "Commit" button to a "Save" button.

5.4 Summary

This chapter has reviewed the major techniques of HCI evaluation. A selection of evaluation techniques has been applied in the evaluation process. The evaluation framework DECIDE was used to guide the whole evaluation procedure. The details of the evaluation study have been described and the results have been recorded and analysed. Conducting the evaluation confirmed that users can use the system to manage UI pattern collections efficiently. In addition, many common problems in the initial prototype were identified and some changes resulting from the evaluation have been made.

Chapter 6 Conclusions and Further Work

In this chapter the research reported in the thesis is reviewed. The contributions of the research are summarised and further work is identified. The research seeks to identify the needs and activities of those developing and consuming pattern collections and to design and implement a prototype to support these ideas and activities.

6.1 Review of the Research

The research for this thesis can be divided into three phases. In the first phase, the context for the study of pattern management tools was identified; the relevant literature and existing tools were reviewed. Based on this work, the research goals were formulated and the research task was established. In the second phase, a new prototype was developed. In the third phase, an evaluation was conducted.

Establishment of the research task

Many HCI patterns and pattern languages or pattern collections have been developed by different researchers (Tidwell, 1999; Borchers, 2001; Welie *et al.*, 2002; Tidwell, 2002; Graham, 2003b; Laakso, 2003; Duyne *et al.*, 2003; Graham, 2003a; Welie, 2004b). These patterns or pattern collections were written in a variety of pattern forms. This raised the issue of how to develop a tool to facilitate researchers and UI designers manipulate pattern collections in a standard pattern structure, PLML.

The motivation for this research included five aspects: (I) identifying the needs and activities from literature to develop pattern collections; (II) evaluating and analysing current UI pattern tools to determine to what extent they met the needs of those manipulating pattern collections; (III) conceptualising a comprehensive pattern management tool based on a standard pattern structure due to the lack of a tool that fully supported PLML; (IV) implementing and evaluating the prototype; and (V) evaluating the PLML specification.

Focused on the research goals, a detailed literature review was conducted. The context and research areas of UI pattern collections were reviewed. Compared to guidelines, the need for

and the advantages of pattern collections were identified. Current methodologies to identify the relationships between pattern and pattern collections were investigated. The various pattern forms were compared and summarised. PLML was evaluated and further enhanced. The review emphasised the important role of forces for pattern selections. Current methodologies related to pattern categorisation were discussed.

The existing specifications for building a pattern management tool were surveyed. The existing pattern tools were divided into three types: catalog-only tool, management tool, and design tool. The common functional features have been identified from this analysis. Furthermore, from the needs identified in the literature and these summarised common features, a comprehensive feature framework (Deng *et al.*, 2005) was proposed to manipulate pattern collections.

Development of the prototype

Deploying the feature framework, a prototype to manipulate pattern collections was conceptualised and developed. The prototype, called MUIP, supported the following main features:

- Pattern authoring based on PLML v1.2.
- Manipulating forces within the system.
- Searching and browsing facilities.
- Pattern modification including pattern versioning and customisation.
- Facility for linking patterns and accumulating the types of relationship.
- Manipulating pattern collections.
- Importing and exporting patterns.

MUIP employed a client-server architecture, which consisted of an end-user client application, a database server and an application running environment. The user can access the central repository to manipulate the UI pattern collections. A group of users can share and manage the same pattern collections from different client computers. Individual users can manipulate the pattern collections on their own computers.

Most but not all of the data the system deals with is pattern related data. The database can:

- Store the data regarding the descriptions of patterns based on PLMLv1.2;

- Include the media data used to illustrate the pattern or address examples of a pattern;
- Record the information about pattern collections;
- Keep track of various versions of patterns;
- Accumulate the types of relationships and store the relationships between patterns;
- Store sets of forces; and
- Support registered users' information.

Evaluation

An evaluation was conducted in order to assess whether the prototype can support users in managing pattern collections efficiently and also to identify the usability problems. The DECIDE framework was selected to guide the evaluation. The results of evaluation confirmed the usefulness of the prototype and the participants used the system to manage UI pattern collections efficiently. Moreover, many common problems in the initial prototype were identified and some changes were made based on the degree of criticality of using the system to manipulate pattern collections.

6.2 Contributions

The major contributions of the research reported in the thesis included the following:

Categorisation of UI pattern tools

It is the first time in the literature that the different types of UI pattern tools have been categorised into three groups.

A *pattern catalog tool* collects and presents patterns organised by different categories; access to patterns via the Internet; and some aid for the creation and submission of new patterns.

A *pattern management tool* provides facilities to manipulate patterns and pattern collections. First, they provide templates for pattern creation. Second, they support searching or browsing collections of patterns and displaying patterns from different views. Third, they provide facilities for modifying existing patterns. Finally, they have mechanisms for

organising patterns and creating relationships between them. These tools emphasise the creation of patterns and managing them within pattern collections.

A *pattern-based design tool* applies patterns to help HCI design activities. Some include basic pattern creation functions as well as customising patterns for an application domain but a number just use a pattern repository without providing any pattern management functions. These tools have automated some processes in user interface design, such as, creating storyboards or generating code. These tools emphasise the use of patterns for designing user interfaces for specific applications.

A comprehensive framework of features

The proposed framework for managing collections of patterns included features identified by the analysis of both the existing specifications and the tools. The features of the framework were grouped into eight functional categories: pattern authoring; manipulating forces; browsing; searching; modification; relating patterns; manipulating collections; input and output. Some features of the framework, such as versioning patterns, manipulating forces, manipulating pattern collections, have not yet been found in other tools. The MUIP system can support these features.

Extending PLML v1.1 to PLML v1.2

The current PLML v1.1 has been refined and extended to produce a new version, PLML v1.2, with more sub-elements, rather than re-ordering the elements. The key reasons for these modifications are presented as follows.

First, in PLML v1.1, only the content contained inside separate tags is searchable and accessible. PLML v1.2 represents forces separately and makes them individually accessible for the purpose of searching and reusing individual forces.

Second, PLML v1.1 does not enable the setting of numerous examples with narrative descriptions and binary image information inside one element. For identifying different objects, PLML v1.2 has appended sub-elements for those elements, such as example, literature, and implementation. Those sub-elements originate from Fincher's discussion (2003) and knowledge of current known patterns.

Third, PLML v1.2 adds a change-log element for patterns because it is necessary to know why and how a pattern is changed. The change-log was organised by date for temporal-based modification activities. The element pattern-link was appended with the revision-number attribute that enables the creation of the relationships between different versions of patterns.

In summary, PLML v1.2 can construct and organise pattern contents more effectively. It has further defined the structure with details for forces, example, literature, and implementation as well as additional change-log data. A comparison with PLMLx indicates that PLML v1.2 is closer to PLML v1.1, so it is easier to map to PLML v1.1.

A pattern management tool overcoming the shortcomings of existing tools

None of the existing tools support all the features summarised in Table 3.4 and none have, as yet, totally implemented PLML. The power of forces has not generally been recognised as an aid for manipulating patterns, exploring the relationships between patterns or selecting patterns to solve UI design problems. None of the existing tools either seemed to support multiple versions of a pattern. Only Damask investigates the customisation of patterns based on the application domain to generate more illustrative examples. Although CoPE allows for the creation and naming of relationship types, the system has no facility to accumulate the link types for future use or research. Most of the tools reviewed do not provide functions for creating new collections. Existing systems do not provide a facility to support mapping the pattern format within the tool with any external resources.

6.3 Further Work

Due to time constraints some features have not been implemented fully. As discussed in the previous section, one of most important features is the visualisation of the relationships between patterns. This can enhance and facilitate the understanding of the inner relationships among patterns or pattern collections. Some other features require to be further enhanced, such as logging. They should be considered and implemented in future work, including:

- The implementation should support the variety of image types and other media data.
- The search function should enable a search based on any subset of the elementary fields. It should provide additional filters to support the search function, as MOUDIL has.

- The visualisation of the relationships between patterns should be implemented.
- This implementation should be enhanced to import and export pattern collections each time, either in PLML formats or an alternative format.
- The administrative function of a database administrator can be added into the system.
- Some features should be further enhanced, such as logging facilities.
- The system should be further optimised to improve the performance and the extensibility.
- The interface requires improvement to be attractive and intuitive. Some interfaces were not always intuitive, because some functionality was hidden and there was a lack of information on the screen.
- MUIP can be extended to enhance other facilities. For example, the support of user specified XML schema for pattern forms should be further integrated into the tool so that the user can do the mapping between various pattern formats and PLML. A mechanism should be added into the system to link the forces with relationships together so that the user can examine the inner meaning of the forces, for each particular relationship, between patterns.
- Furthermore, the forces should further be used to identify the relationships between patterns or pattern collections. Further UI pattern conceptual study based on the forces is an interesting topic.

In addition, by manipulating UI pattern collections using MUIP, this process aids researchers or UI designers in accumulating the evidence to evaluate PLML v1.1 and PLMLv1.2 for further improvement. MUIP can act as the base to build a pattern-based design tool, such as supporting code generation based on a further formalised PLML.

Appendix A: PLML v1.1

PLML v1.1

```
<!ELEMENT pattern (name?, alias*, illustration?, problem?,
context?, forces?, solution?, synopsis?, diagram?, evidence?,
confidence?, literature?, implementation?, related-patterns?, pattern-link*, management?)>
 <!ATTLIST pattern patternID CDATA #REQUIRED>
 <!ELEMENT name (#PCDATA)>
 <!ELEMENT alias (#PCDATA)>
 <!ELEMENT illustration ANY>
 <!ELEMENT problem (#PCDATA)>
 <!ELEMENT context ANY>
 <!ELEMENT forces ANY>
 <!ELEMENT solution ANY>
 <!ELEMENT synopsis (#PCDATA)>
 <!ELEMENT diagram ANY>
 <!ELEMENT evidence (example*, rationale?)>
 <!ELEMENT example ANY>
 <!ELEMENT rationale ANY>
 <!ELEMENT confidence (#PCDATA)>
 <!ELEMENT literature ANY>
 <!ELEMENT implementation ANY>
 <!ELEMENT related-patterns ANY>
 <!ELEMENT pattern-link EMPTY>
 <!ATTLIST pattern-link
   type CDATA #REQUIRED
   patternID CDATA #REQUIRED
   collection CDATA #REQUIRED
   label CDATA #REQUIRED
 >
 <!ELEMENT management (author?, revision-number?, creation-date?, last-modified?, credits?)>
 <!ELEMENT author (#PCDATA)>
 <!ELEMENT creation-date (#PCDATA)>
 <!ELEMENT credits (#PCDATA)>
 <!ELEMENT revision-number (#PCDATA)>
 <!ELEMENT last-modified (#PCDATA)>
```

Appendix B: PLML v1.2

PLML v1.2

```
<!ELEMENT pattern (name?, alias*, illustration?, problem?, context?, forces?, solution?,
synopsis?, diagram?, evidence?, confidence?, literature?, implementation*, related-
patterns?, pattern-link*, management?)>
<!ATTLIST pattern patternID CDATA #REQUIRED collection CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
<!ELEMENT alias (#PCDATA)>
<!ELEMENT illustration ANY>
<!ELEMENT problem (#PCDATA)>
<!ELEMENT context ANY>
<!ELEMENT forces (force *)>
<!ELEMENT force ANY>
<!ELEMENT solution ANY>
<!ELEMENT synopsis (#PCDATA)>
<!ELEMENT diagram ANY>
<!ELEMENT evidence (example*, rationale?)>
<!ELEMENT example (example-name?,example-diagram?,description?,knownuses?)>
<!ELEMENT example-name (#PCDATA)>
<!ELEMENT example-diagram ANY>
<!ELEMENT description ANY>
<!ELEMENT knownuses ANY>
<!ELEMENT rationale ANY>
<!ELEMENT confidence (#PCDATA)>
<!ELEMENT literature (workname?,reference?)>
<!ELEMENT workname (#PCDATA)>
<!ELEMENT reference (#PCDATA)>
<!ELEMENT implementation (implementation-name?,code?,otherdetails?)>
<!ELEMENT implementation-name (#PCDATA)>
<!ELEMENT code ANY>
<!ELEMENT otherdetails ANY>
<!ELEMENT related-patterns ANY>
<!ELEMENT pattern-link EMPTY>
<!ATTLIST pattern-link  type CDATA #REQUIRED
                        patternID CDATA #REQUIRED
                        collection CDATA #REQUIRED
                        revision-number CDATA #REQUIRED
                        label CDATA #REQUIRED>
<!ELEMENT management (author?, credits?, creation-date?, last-modified?, revision-
number?, change-log*)>
<!ELEMENT author (#PCDATA)>
<!ELEMENT credits (#PCDATA)>
<!ELEMENT creation-date (#PCDATA)>
<!ELEMENT last-modified (#PCDATA)>
<!ELEMENT revision-number (#PCDATA)>
<!ELEMENT change-log (log-creation-date?, log-content?)>
<!ELEMENT log-creation-date (#PCDATA)>
<!ELEMENT log-content ANY>
```

Appendix C: Use Case Specifications and Main Scenarios

Use Case Specifications

1. Register User

Brief Description

User registers as a legal user of the system.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (*Open DataBase Connectivity*) that sends changes to database.

Pre-conditions

The User executes the system and then the User is viewing the "Login" window.

Basic Flow

1. User selects "Register" button
2. System shall display the "Register" window
3. User enters the "First Name" and "Last Name"
4. User selects "Save" button
5. Data service shall save user's information into database
6. System shall closes the "Register" window
7. System shall display user login information in the "Login" window

Post-conditions

1. System displays user login information in the "User name" field

Alternative Flows

User Selects "Cancel" button

Branches from basic flow 3

1. User selects "Cancel" button
2. System shall close the "Register" window

2. Login

Brief Description

User logs on the system.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***Open DataBase Connectivity***) that sends changes to database.

Pre-conditions

The User executes the system and then the User is viewing the "Login" window.

Basic Flow

1. User selects "User name"
2. User selects "Login" button
3. Data Service shall validate the user information
4. System shall display the MUIP main window
5. System shall display user login information in the MUIP main window

Post-conditions

1. System displays MUIP main window with user login information

Alternative Flows

User Does Not Select User Name

Branches from basic flow 1

1. User selects "Login" button
2. System shall display message to request user to select "User name"
3. Branch to basic flow 1

User Selects "Cancel" Button

Branches from basic flow 1

1. User selects "Cancel" button
2. System shall close the "Login" window

3. Find Existing Pattern

Brief Description

User searches for a pattern for seeing the details of the target pattern.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***Open DataBase Connectivity***) that sends changes to database.

Pre-conditions

The User logins the system and then the User is viewing the "Pattern Finder".

Basic Flow

1. User enters search criteria
2. User selects "Search"
3. System shall validate data
4. System shall send request to Data Service

5. Data Service shall return results
6. System shall display results in the "Search Results" field
7. User selects result and then "View"
8. System shall send request to Data Service
9. Data Service shall return data
10. System shall display pattern data in the "Pattern Detail" section

Post-conditions

1. System displays results in the "Search Result" field
2. System clears search criteria

Alternative Flows

System Cannot Validate "Pattern ID" Data

Branches from basic flow 2

1. System shall pop-up message request user to enter correct data for "Pattern ID"
2. User selects "OK" in the pop-up window
3. System shall clear the "Pattern ID"
4. User enters data for "Pattern ID"
5. Branch to basic flow 2

User Doesn't Enter Any Search Data

Branches from basic flow 2

1. System shall display message to request User to enter search data
2. User selects "OK" in the pop-up window
3. System shall display no results in the "Search Results"
4. Branch to basic flow 1

Data Service Returns Empty Result

Branches from basic flow 5

1. System shall display no results in the "Search Results"
2. Branch to basic flow 1

User Clicks Any Column-header of "Search Results"

Branches from basic flow 6

1. User clicks any column-header of "Search Results"
2. System shall refresh view and display the reordered searching results based on the column-header
3. Branch to basic flow 7

Double Click a Result from the List

Branches from basic flow 7

1. User double clicks a Customer or Advertiser from list
2. Branch to basic flow 8

4. Create New Pattern

Brief Description

User creates a new Pattern in the "Pattern Detail" section.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The User is viewing the "Pattern Detail" section.

Basic Flow

1. User selects "New"
2. System shall refresh view and display blank record in the different tabs of the "Pattern Detail" section
3. System shall display the first tab from the "Pattern Detail" section
4. User edits pattern data in the first tab (Refer to Use Cases: Assign Pattern to a Collection, Create Alias, Create Management, Add an Illustration)
5. User selects "Commit" in this tab
6. System sends request to Data Service
7. Data Service shall save pattern data into the database
8. Data Service shall return a "Pattern ID" for the new pattern
9. System shall display the ID in the "Pattern ID" field
10. User selects other tabs and edits pattern data (Refer to Use Cases: Add other Data, Add a Diagram, Create Literature, Create Evidence, Create Force List, Create Implementation)
11. User selects the "Commit All" above the tabs
12. Data Service shall save all pattern data into the database

Post-conditions

1. System creates and saves a new pattern into the database

Alternative Flows

User Doesn't Enter Pattern Name

Branches from basic flow 5

1. System shall display message to request User to enter Pattern Name
2. User selects "OK" in the pop-up window
3. Branch to basic flow 4

User Selects "Commit All"

Branch from basic flow 5

1. User selects the "Commit All"
2. Branch to basic flow 6

User Doesn't Select "Commit"s or "Commit All"

Branch from basic flow 5, 11

1. User forgets to select "Commit"s or "Commit All"
2. System will not save the new pattern into the database

5. Manipulate Forces

Brief Description

User manipulates forces.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The User is viewing Forces window.

Basic Flow

1. User enters "Force", "Other details" and "Label" data
2. User selects the "New" button.
3. System shall send request to Data Service
4. Data Service shall save the new "Force" data into database
5. System will display the new force in the "All forces" list

Post-conditions

1. System will display the new force in the "All forces" list

Alternative Flows

User updates the selected force

Branches from basic flow 1

1. User edits the related data of the selected force
2. User selects "Update" button
3. System shall send request to Data Service
4. Data Service shall save the "Force" data into database

User searches patterns based on force

Branches from basic flow 1

1. User enters the force in the "Force" search field
2. User selects "Search" button
3. System shall send request to Data Service
4. Data Service shall return the search results
5. System displays the search results in the "Search Results(patterns)" list

6. User selects one pattern from the list
7. System displays all forces of the selected pattern in the "Force List"

User views all forces of a collection

Branches from basic flow 1

1. User selects a collection name from the "Collection" field
2. System shall send request to Data Service
3. Data Service shall forces belonged to the selected pattern collection
4. System displays all forces belonged to the selected pattern collection in the "Force List"

6. Browse Pattern

Brief Description

User browses the detail of a selected pattern.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The user is viewing a pattern list.

Basic Flow

1. User selects a pattern from the list.
2. System shall send request to Data Service
3. Data Service shall returns the details of the selected pattern
4. System displays the details of the selected pattern.

Post-conditions

1. System displays the details of the selected pattern.

7. Modify Pattern

Brief Description

User modifies the detail of the pattern.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The user is viewing a pattern in the pattern editor.

Basic Flow

1. User modifies the piece of data of the pattern

2. User selects "Commit All" button
3. System shall send request to Data Service
4. Data Service shall save the changed data of the selected pattern into the database

Post-conditions

1. Data Service shall save the changed data of the selected pattern into the database

8. Link Pattern

Brief Description

The User creates relationship between related patterns.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Pre-conditions

The User is viewing a pattern in the "Related Pattern &Log" tab of the "Pattern Detail" section. Patterns are already stored in the pattern repository.

Basic Flow

1. User selects a related "Pattern ID" from the List
2. System shall display the "Name" and "Collection"
3. User selects "Pattern Link Type" from the List
4. User edits the "Label"
5. User selects "Add"
6. System shall display them in a new record in the "Related Pattern List"
7. User selects an item from the "Related Pattern List"
8. System shall display "Pattern ID" ,"Pattern Link Type", "Collection" and "Label" of the selected item in the appropriate fields and highlights the selected item
9. User modifies above fields
10. User selects "Edit"
11. System shall display the updated item in the List

Post-conditions

1. System displays "Related Pattern List"

Alternative Flows

User Deletes a Relationship

Branches from basic flow 7

1. User selects "Delete"
2. System shall pop-up message to confirm the operation
3. User selects "Yes"
4. System shall delete the selected item from the List
5. Branch to basic flow 11

User Creates New Link Type

Branches from basic flow 3

1. Refer to User Case: Create Link Type

9. Manipulate Collection

Brief Description

User manipulates pattern collections.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (*Open DataBase Connectivity*) that sends changes to database.

Pre-conditions

The User is viewing Collection windows.

Basic Flow

1. User enters a collection name in the "Name" field
2. User enters data for the "Editor" and "Description" fields
3. User selects the "New" button
4. System shall send request to Data Service
5. Data Service shall save the new "Collection" data into database

Post-conditions

1. System saves the new "Collection" data into database

Alternative Flows

User modifies the details of a collection

Branches from basic flow 1

1. User edits the related data of the selected collection.
2. User selects "Update" button
3. System shall send request to Data Service
4. Data Service shall save the "Collection" data into database

User collects a pattern into a collection

Branches from basic flow 1

1. User selects a collection name from the "Name"
2. User selects a pattern item from the "Pattern in the system" list
3. User selects the "Collect" button
4. System shall send request to Data Service
5. Data Service shall assign and create a pattern and save the pattern into database as a member of the selected pattern collection.

10. Import Pattern

Brief Description

The User imports a pattern written in different formatted file.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Pre-conditions

The User logins the system.

Basic Flow

1. User selects "Import"
2. System shall display "PLML" tab in Pattern Detail section
3. System shall clears all fields from in the Pattern Detail section
4. User imports a pattern written in PLML v1.2 (Refer to Use Case: Import PLML)
5. User imports a pattern written in alternative formats (Refer to Use Case: Import Alternatively Formatted Patterns)
6. System shall record all contents of the pattern in appropriate fields

Post-conditions

1. System records all contents of the pattern in appropriate fields

11. Export Pattern

Brief Description

The User document a pattern in a file.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The User is viewing any version of a pattern in the "Pattern Detail" section.

Basic Flow

1. User selects "PLML" tab
2. User exports a pattern in PLML (Refer to Use Case: Export PLML vx.x)
3. User exports a pattern in alternative formatted file (Refer to Use case: Export Pattern in Alternative Format)
4. System shall save the pattern in target file

Post-conditions

1. System saves the pattern in target file

12. Create New Version

Brief Description

The User makes modification for an existing pattern and wants to save as a new version.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (*Open DataBase Connectivity*) that sends changes to database.

Pre-conditions

The User is viewing or editing a pattern in the first tab of the "Pattern Detail" section. The pattern has at least one version number.

Basic Flow

1. User modifies the existing pattern in different tab
2. User enters a new version number in the "Revision Number"
3. User clicks other filed or type "Tab" in the keyboard
4. System shall pop-up message to request user to confirm to save previous version
5. User selects "Yes" in the pop-up message
6. System keeps track of previous version
7. System sends a request to Data Service
8. Data Service saves new version of pattern into database
9. Data Service shall return the all version list
10. System shall display the version list in the "Revision Number"

Post-conditions

1. System creates a new version for the pattern in the database

Alternative Flows

User Selects "No"

Branches from basic flow 5

1. User selects "No" in the pop-up message
2. System shall not keep previous version for the pattern
3. System shall change the revision number for the pattern

User Selects New Version from the main menu

Branches from basic flow 3

1. User selects the "New Version" from the main menu
2. Branch to basic flow 4

User Selects a Revision Number

Branches from basic flow 10

1. User selects a revision number from the List

2. System shall pop-up to request user to confirm to display those data of the pattern in that version
3. Users selects "Yes" in the pop-up message
4. System shall display the data of the pattern in the selected version

13. Customise Pattern

Brief Description

The User makes modification for an existing pattern based on specific design problem.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The User is viewing or editing a pattern in the first tab of the "Pattern Detail" section. The pattern has at least one version number. User is viewing the latest version of the pattern.

Basic Flow

1. User selects different tab in the Pattern Detail section.
2. User modifies the existing pattern in different tab according to current design problem
3. User selects "Commit All"
4. System shall send a request to Data Service
5. Data Service saves the change into the database

Post-conditions

1. System saves the changes into database

Alternative Flows

User Selects "Commit" in each tab

Branches from basic flow 3

1. User selects "Commit" in each tab
2. Branch to basic flow 4

14. Write Comments

Brief Description

The User writes comments for modification or reviewing a pattern.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Pre-conditions

The User is viewing a pattern in the "Related Pattern&Log" tab of the "Pattern Detail" section.

Basic Flow

1. User selects current date from the "Log List"
2. System shall display the "Log Content" and highlight the selected item
3. User adds comments into the "Log Content"
4. User selects "Edit"
5. System shall record "Log Content" inside the selected item in the "Log List"
6. System shall display the updated item in the List

Post-conditions

1. System displays "Log List"

Alternative Flows

No Current Date's Record

Branches from basic flow 1

1. User edits comments in "Log Content"
2. User selects "Add"
3. System shall display the new item for current date in "Log List"
4. Branch to basic flow 6

User Selects "Delete"

Branches from basic flow 4

1. User selects "Delete"
2. System shall delete the selected item from the List
3. Branch to basic flow 6

15. Create Link Type

Brief Description

The User creates new relationship types.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The User is viewing relationship Type window.

Basic Flow

1. User enters new name of relationship type into the "Type" field
2. User enters "User", "Proposers" and "Description" data
3. User selects the "New" button
4. System shall send a request to Data Service
5. Data Service creates a new relationship type and saves the change into the database

6. Data Service returns a list of the updated relationship types
7. System shall display the updated type items in the Type List

Post-conditions

1. System displays the updated list in the "Pattern Link Type"

16. Import PLML

Brief Description

The User imports a pattern written in PLML v1.2.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Pre-conditions

The User is viewing the "PLML" tab of the "Pattern Detail" section.

Basic Flow

1. User selects "Import" inside the "PLML" field
2. System shall display a open file window
3. User selects a pattern written in PLML v1.2
4. User selects "Open"
5. System shall display the PLML file in the PLML field
6. System shall parse the PLML and put all pattern contents into appropriate fields

Post-conditions

1. System displays pattern in the appropriate fields

17. Import Alternatively Formatted Pattern

Brief Description

The User imports a pattern written in RTF, TXT, HTML, XML.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Pre-conditions

The User is viewing the "PLML" tab of the "Pattern Detail" section.

Basic Flow

1. User selects "Import" inside the "Pattern" field
2. System shall display a open file window
3. User selects a pattern written in RTF or TXT or HTML
4. User selects "Open"
5. System shall display the pattern file in the "Pattern" field
6. User selects some part of content of the pattern

7. User clicks the right key of mouse
8. System shall display a context menu
9. User selects the name for the selected content
10. System shall paste the selected content into the appropriate fields
11. System shall generate a PLML in "PLML" field with the focus on the result of latest operation
12. User selects "Undo"
13. System shall cancel the previous step operation

Post-conditions

1. System displays contents of pattern in the appropriate fields

18. Export PLML v x.x

Brief Description

The User document a pattern in PLML.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***O**pen **D**ata**B**ase **C**onnectivity*) that sends changes to database.

Pre-conditions

The User is viewing any version of a pattern in the "Pattern Detail" section.

Basic Flow

1. User selects "PLML" tab
2. User selects a PLML version number
3. User selects "Create" in "PLML" field
4. System shall send a request to Data Service
5. Data Service shall retrieve the specific version of pattern data from database
6. Data Service shall return the result
7. System shall get all content data of the pattern from Data Service and generate PLML
8. System shall display the PLML in the "PLML" field
9. User selects "Save File"
10. System shall display save file window
11. User selects target folder and edit the name for the file
12. User selects "Save"
13. System shall create a new file in the target folder

Post-conditions

1. System displays the PLML in the "PLML" field and saves a new PLML file in target folder

Alternative Flows

User selects "Includes image data"

Branches from basic flow 3

1. User selects "Yes" from "Includes Diagram/Pictures" field
2. Branch to basic flow 3

19. Export Pattern in Alternative Format

Brief Description

The User documents a pattern in narrative description in a text file.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Data Service

The Data Service is the ODBC (***Open DataBase Connectivity***) that sends changes to database.

Pre-conditions

The User is viewing any version of a pattern in the "Pattern Detail" section.

Basic Flow

1. User selects "PLML" tab
2. User selects "Create" in "Pattern" field
3. System shall send a request to Data Service
4. Data Service shall retrieve the specific version of pattern data from database
5. Data Service shall return the result
6. System shall get all content data of the pattern from Data Service and generate narrative description
7. System shall display the descriptive pattern in the "Pattern" field
8. User selects "Save File"
9. System shall display save file window
10. User selects target folder and edit the name for the file
11. User selects "Save"
12. System shall create a new file in the target folder

Post-conditions

1. System displays the narrative pattern in the "Pattern" field and saves a new pattern file in target folder

Alternative Flows

The Pattern Includes Images

Branches from basic flow 8

1. If the descriptive pattern includes metafile data
2. User selects "*.rtf" or "*.Doc" type

3. User edits a file name
4. Branch to basic flow 11

The Pattern Includes no Images

Branches from basic flow 8

1. If the descriptive pattern includes no metafile data
2. User selects any file type
3. User edits a file name
4. Branch to basic flow 11

20. Maintain Database

Brief Description

The User of admin level administrates the database.

Actors

User

The User is the person who uses MUIP.

System

The System is MUIP.

Pre-conditions

The user is viewing the data tables of the back end database.

Basic Flow

1. User performs the maintenance functionality such as insertion, deletion, updating of data
2. Data Service shall saves the change of the data into database

Post-conditions

1. Data Service shall save the change of the data into database

Main Specific Scenarios

Scenario 1:

Context: A student David has got a collection of Mary Zajicek's patterns. He has saved each pattern from the collection in a TXT file in a specific folder "Mary". He wants to store those patterns into the MUIP system quickly.

Flow of events:

1. David logins the MUIP system.
2. David clicks "Import" button in the system's interface.
3. System displays the PLML tab in Pattern Detail Section.
4. David clicks the "Import" button in the Pattern field.
5. System displays an open file window.
6. David uses the open file window to navigate his computer file system to locate the "Mary" folder.
7. David chooses one TXT file from the folder.
8. David selects the "Open" button in the window.

9. System closes the open file window and displays the data from the selected TXT file in the textbox in Pattern field.
10. David selects Pattern ID from the textbox and clicks the right button of the mouse.
11. System displays a context menu which includes all content names of a pattern such as: Pattern ID, Problem, Collection, Illustration, Context, Solution, etc.
12. David selects the Pattern Name from the context menu by clicking.
13. System copies and pastes the Pattern Name into appropriate field and generates a PLML which includes the Pattern Name data.
14. The PLML is displayed in the PLML field which highlights the Pattern Name data.
15. David repeats the steps from step 10 to step 14, he can transfer all contents of the pattern into the appropriate fields in the system.
16. In current operation, David finds that he make a mistake to paste "Problem" content into "Context" field by watching the generated PLML.
17. David clicks the "Undo" button.
18. System cancels last operation and removes "Problem" content from the "Context" field. It refreshes the result in the PLML as well.
19. David has transferred all content of the pattern and he clicks the "Commit All" button.
20. System saves all the contents of the pattern into the repository.
21. System displays the Pattern ID in the appropriate field.
22. David wants to import other patterns in the collection as well.
23. David repeats the step from step 4 to step 21 for importing each pattern.

Scenario 2:

Context: A student David remembers he has imported a collection of Mary Zajicek's patterns into the MUIP system. He knows those patterns are created based on a set of forces. He wants to know which patterns solve the same forces in the collection. But he cannot remember the pattern names and specific forces.

Flow of events:

1. David logins the MUIP system.
2. David enters the "Mary" in the "Author" and "Collection" fields in Pattern Finder section.
3. David clicks the "Search" button.
4. System displays all related patterns in the Search Results, which are created by the author name similar to "Mary" and organised in the collection name similar to "Mary".
5. System displays each pattern which includes ID, Name, Author and Problem.
6. David finds all patterns in the list from Mary Zajicek.
7. David double clicks on one pattern in the list.
8. System displays the contents of the selected pattern in the Pattern Detail section.
9. David selects the "Problem" tab in the Pattern Detail section.
10. System displays forces list in the tab.
11. David selects each force from the list.
12. System displays the detailed content of the selected force in the below textbox.
13. After scanning those forces for the pattern, David wants to know other patterns which deal with the same force and wants to see the difference between them by seeing the detail of them.

14. David copies one force from the List and pastes into the "Force" field in the Pattern Finder section.
15. David clicks the "Search" button.
16. System displays a list of patterns which solve the same force.
17. David can see the details of those patterns by selecting them one by one from the list.
18. David can browse the same force in different patterns and see what are the relationships between the same force with other forces in different patterns.
19. Based on the relationships between the forces from different patterns, David can further to judge and classify the relationship between those patterns.

Scenario 3:

Context: A student James wants to search a pattern for solving his current design problem. Once he can apply the pattern in implementing his current design. He wants to save his implementation code and adds additional information to explain how to apply this pattern in design. He also wants to make a version of this pattern focusing on his current design problem.

Flow of events:

1. James logins the MUIP system.
2. James enters possible pattern name, problem description in the searching field of Pattern Finder section.
3. James selects the "Search" button.
4. System returns a list of results.
5. James finds there is no the pattern he wants.
6. James does the search again by typing different words.
7. James finds the target pattern in the list.
8. James selects the target pattern and sees the details of the pattern.
9. James applies the pattern to solve his current design problem.
10. James modifies the pattern according to current design.
11. James changes the description of problem and context based on the current design problem.
12. James searches for some forces.
13. System displays no search results.
14. James selects "Show all forces from database".
15. James selects some specific forces and adds to force list.
16. James creates some specific forces due to current design and adds to force list.
17. James adds the implementation code and explanations regarding to the implementation.
18. James gives a new version name to the pattern.
19. System creates change comment for this change.
20. System request James to backup previous version of the pattern.
21. James chooses to backup previous version.
22. System save previous version.
23. James selects "Commit All".
24. System saves the new version of the pattern.

Scenario 4:

Context: A small group of students want to find and apply a collection of patterns to solve a current complex design problem. They divide the complex problem into small design

problems and assign to each one to find appropriate patterns to solve it. Individual will assign a specific collection with agreed name to the found pattern and other can browse the collection name for gathering patterns to solve the design problem.

Flow of events:

1. Group member A (GMA) logins the system.
2. GMA searches pattern for the small design problem.
3. GMA finds target pattern for solving the small design problem.
4. GMA adds comments or explanations to explain how the pattern can apply to solve the small design problem.
5. GMA assigns the pattern into the agreed collection name.
6. GMA selects "Commit All".
7. System saves the changes.
8. Other group members search patterns for the other small design problems by repeating the step 1 to step 6.
9. GMA searches related patterns for the complex design problem by searching the agreed collection name.
10. GMA sees the details of those patterns by selecting them one by one.
11. System displays the selected pattern in detail.
12. GMA sees the comments and changes of the selected pattern.
13. GMA can give his/her comments and make changes to those patterns selected by other group members.
14. GMA selects "Commit All".
15. System saves the changes and comments.
16. Other group members can browse and see the comments and changes for those patterns in the specific collection.

Scenario 5:

Context: A student Jim wants to create relationship between patterns. Especially, he is very interested in the Pattern A (PA). He wants to find out the related patterns to PA and create relationships between them.

Flow of events:

1. Jim logins the system.
2. Jim enters the pattern name and alias in the appropriate fields in Pattern Finder section.
3. Jim selects "Search".
4. System returns the search results.
5. Jim can see those patterns in details which have the similar name to PA.
6. Jim enters PA's force in the field of Pattern Finder section.
7. Jim selects "Search".
8. System displays patterns which are related to the force.
9. Jim selects those patterns one by one and sees the detail of them.
10. Jim analyses the relationship between those patterns.
11. Jim searches PA and views it in detail.
12. Jim selects the "Related Patterns &Log" tab.
13. Jim selects an existing pattern from the Pattern ID list.
14. System displays the name and collection of this selected pattern.
15. Jim creates a new relationship types in the "Pattern Link Type" fields.
16. Jim clicks the "Add".
17. System displays the relationship in the "Pattern Link List".

18. Jim repeats the steps from step 13 to step 17 and creates those relationships between related patterns.
19. Jim selects "Commit All".
20. System saves those relationships into database.
21. System displays the new relationship types in the "Pattern Link Type" field.

Scenario 6:

Context: A student Jim wants to export the version 2 of a pattern in PLML v1.2, which includes image data.

Flow of events:

1. Jim logins the system.
2. Jim enters the pattern name in the appropriate fields in Pattern Finder section.
3. Jim selects "Search".
4. System displays the search results.
5. Jim selects the pattern name in the search results.
6. Jim clicks "View".
7. System displays the selected pattern in the Pattern Detail section.
8. Jim selects the Version 2 of the pattern.
9. System displays the Version 2 of the pattern.
10. Jim selects the PLML tab.
11. Jim selects "Yes" from "Includes Diagram/Pictures" field.
12. Jim clicks the "Create" button.
13. System displays the generated PLML in the appropriate field.
14. Jim clicks "Save File" button.
15. System display save file window.
16. Jim chooses target folder.
17. Jim enters file name.
18. Jim clicks the "Save" button.
19. System saves PLML in the target folder.

Appendix D: Database Schema in SQL Statement

muip.sql

```
CREATE DATABASE MUIP;

USE MUIP;

CREATE TABLE Alias (
        AID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        PID INT NOT NULL ,
        Alias VARCHAR(50) NULL,
        PRIMARY KEY(AID)
);

CREATE TABLE Example (
        EID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        PID INT NOT NULL ,
        Name VARCHAR (50) NULL ,
        Picture MEDIUMBLOB NULL ,
        Description MEDIUMTEXT NULL ,
        KnowUses MEDIUMTEXT  NULL ,
        PRIMARY KEY(EID)
);

CREATE TABLE Forces (
        FID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        Forcea MEDIUMTEXT NULL ,
        PID INT NULL,
        PRIMARY KEY(FID)
);

CREATE TABLE Implementation (
        IID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        PID INT NOT NULL ,
        Name VARCHAR (200) NULL ,
        Code MEDIUMTEXT NULL ,
        OtherDetails MEDIUMTEXT NULL ,
        PRIMARY KEY(IID)
);

CREATE TABLE Literature (
        LID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        PID INT NOT NULL ,
        WorkName VARCHAR (200) NULL ,
        Reference MEDIUMTEXT NULL ,
        PRIMARY KEY(LID)
);

CREATE TABLE Log (
        LOGID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
```

```
        PID INT NOT NULL ,
        CDate DATE NULL ,
        ChangeLog MEDIUMTEXT NULL ,
        PRIMARY KEY(LOGID)
);

CREATE TABLE Management (
        ManagementID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        PID INT NOT NULL ,
        Authors VARCHAR (50) NULL ,
        Credits VARCHAR (50) NULL ,
        CreationDate DATE NULL ,
        LastModifiedDate DATE NULL ,
        PRIMARY KEY(ManagementID)
);

CREATE TABLE OurUsers (
        UID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        FirstName VARCHAR (50) NOT NULL ,
        LastName VARCHAR (50) NOT NULL ,
        PassWord VARCHAR (50) NULL ,
        PRIMARY KEY(UID)
);

CREATE TABLE Patterns (
        PID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        PatternID INT NOT NULL ,
        CollectionID INT NOT NULL ,
        CollectionPID INT NULL ,
        VersionNumber VARCHAR (50) NULL,
        PatternName VARCHAR (50) NULL ,
        Illustration MEDIUMBLOB NULL ,
        Problem MEDIUMTEXT NULL ,
        Context MEDIUMTEXT NULL ,
        Solution MEDIUMTEXT NULL ,
        Synopsis MEDIUMTEXT NULL ,
        Diagram MEDIUMBLOB NULL ,
        Rationale MEDIUMTEXT NULL ,
        Confidence VARCHAR (50) NULL ,
        Author VARCHAR (50) NULL ,
        RelatedPatterns MEDIUMTEXT NULL ,
        PRIMARY KEY(PID)
);

CREATE TABLE RelatedPatterns (
        RID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        PID INT NOT NULL ,
        RelatedPID INT NULL ,
        TypeID INT NOT NULL ,
        Label VARCHAR (50) NULL ,
        PRIMARY KEY(RID)
);
```

```
CREATE TABLE Type (
        TypeID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        Type VARCHAR (50) NULL ,
        Description VARCHAR (200) NULL ,
        RelatedPID INT NULL ,
        UID INT NULL ,
        PRIMARY KEY(TypeID)
);

CREATE TABLE Collection(
        CollectionID INT DEFAULT 0 AUTO_INCREMENT NOT NULL ,
        CName VARCHAR (50) NULL ,
        CDescription VARCHAR (200) NULL ,
        CreationDate DATE NULL ,
        Editor VARCHAR (50) NULL ,
        PRIMARY KEY(CollectionID)
);

use mysql;
Insert Into user (host,user,password) Values('Localhost','jdeng',password('jim'));
Insert Into
db(host,db,user,Select_priv,Insert_priv,Update_priv,Delete_priv,Create_priv,Drop_priv)
Values('Localhost','muip','jdeng','Y','Y','Y','Y','Y','Y');
Flush PRIVILEGES;
```

Appendix E: Handy Hints for Evaluating the MUIP

Handy Hints

Searching Patterns

1. Enter searching data into different field and click the "Search" button
2. Double click on a target pattern in the "Search Results" and the detail will be displayed in the below "Pattern Detail Section" which is below the "Pattern Finder" section. Click right button of mouse above a target pattern, the detail of the pattern will be shown in another window.

Creating new pattern

1. Click the "New" button and enter all data into appropriate fields
2. Click "Commit All" button or "Commit" button into different tabs.
3. A new button at least has "Name", "Collection name", "Revision Number".

Creating relationship type

1. Click the "Relationship type" from the Main menu and another "Type" widow will be displayed.
2. Edit appropriate data for a new relationship type in the "Type" window.
3. Click the "New" button to save the new relation type.
4. Modify data for an existing relationship type and click "Update" button to save the change.
5. Click "Exit" to close the "Type" window

Creating a new version for a pattern

1. View the contents of a selected pattern in the "Pattern Detail Section".
2. Enter a new version number by using "0~9" and "." into the "Revision Number" field.
3. Click the "New version" in the Main Menu.
4. Click "Yes" to create and save a new version. Click "No" to cancel the operation.

Creating a new collection

1. Click "Collection" in the Main Menu or double click the "Collection" field in the "Pattern Detail Section". Another "Collection" window will be shown.
2. Edit collection "Name", "Editor", "Description".
3. Click "New" to create and save a new collection called as the data from the "Name" field.
4. Select the collection name from the "Name" filed and see the detail of the collection.
5. Modify an existing collection data and click "Update" to save the change.

Collecting a pattern into a collection

1. Open the "Collection" window
2. Select the collection from the "Name" field.
3. Select a pattern from the "Patterns in system" list by clicking the head of the target row.
4. Click the "Collect" to copy the pattern into the selected collection.

Importing pattern in alternative formats such as RTF, TXT, XML not PLML v1.2,etc.

1. Click the "PLML" tab in the "Pattern Detail Section" and the tab will be displayed.
2. Click the "Import" button in the "Pattern" sub-section and another "Open file dialog" will be displayed.
3. Open a RTF file or other supporting format file and the content of the selected file will be displayed in the "Pattern" textbox.
4. Select information data without heading.
5. Click the right key of mouse to show the Context Menu and select appropriate option in the Context Menu.
6. Check previous operations and use "Undo" to cancel previous wrong operations.
7. Repeat step 4-6 until transferring all information data into appropriate fields.

Copy and Paste

1. Copy: "Ctrl+ c"
2. Paste: "Ctrl+v"

Clear all fields

1. Click the "New pattern" in the Main Menu

Appendix F: Information Sheet for Evaluating the MUIP

MUIP Evaluation

Student

Junhua Deng
Email: J.Deng@massey.ac.nz
Phone: 3505799, ext. 7469
Room AHA 3.73

Supervisors:

A/P Elizabeth Kemp
Email e.kemp@massey.ac.nz
Phone 3505799, ext. 2469
Room AHA 3.87

Mrs Elisabeth-Ann Todd
Email E.Todd@massey.ac.nz
Phone 350 5799 ext. **2455**
Room AHA 3.84

The purpose of the research is to evaluate the usability and functionality of MUIP, a pattern management system developed to manipulate existing user interface pattern collections.

This project has been evaluated by peer review and judged to be low risk. Consequently, it has not been reviewed by one of the University's Human Ethics Committees. The researcher(s) named above are responsible for the ethical conduct of this research.

If you have any concerns about the conduct of this research that you wish to raise with someone other than the researcher(s), please contact Professor Sylvia Rumball, Assistant to the Vice-Chancellor (Ethics & Equity), telephone 06 350 5249, email humanethicspn@massey.ac.nz.

Completion and return of the questionnaire implies consent. You have the right to decline to answer any particular question.

Appendix G: Informed Consent Form for MUIP Evaluation

Informed Consent Form

I state that I am over 18 years of age and wish to participate in a program of research being conducted by Mr. Jim Deng and his supervisors at the Institute of Information Sciences and Technology, Massey University.

The purpose of the research is to evaluate the usability and functionality of the MUIP, a pattern management system developed to manipulate existing user interface pattern collections.

The procedures involve the monitored use of MUIP. I will be asked to perform specific tasks using MUIP. As I complete some tasks, I will be observed. Observation data includes handwriting notes. I will also be asked open-ended questions about the MUIP and my experience using it.

All information collected in the study is confidential, and my name will not be identified at any time.

I understand that I am free to ask questions or to withdraw from participation at any time without penalty.

Signature: _____ **Date:** _____

(Adapted from Cogdill,1999)

Appendix H: Observation Form (observational protocol) for MUIP Evaluation

Observation Form for MUIP Usability Evaluation (protocol)

Participant No. _____

Scenario 1:

Time Started: _____ Time Completed: _____

Total number of slips: _____

Total number of errors: _____

Main tasks:

Main tasks	Action	Notes
Searching collection	1. Use help to find out what to do (choose one)? o Yes o No 2. How many times for help? Why? 3. Does it find the all patterns from the colleciton successfully? o Yes o No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Displaying a selected pattern content in current interface (select and click the "view")	1. Use help to find out what to do (choose one)? o Yes o No 2. How many times for help? Why? 3. Does it display the content of target pattern successfully? o Yes o No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Browsing pattern content especially in the forces	1. Use help to find out what to do (choose one)? o Yes o No 2. How many times for help? Why? 3. Does it browse the forces of target pattern successfully? o Yes o No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Search and view target force	1. Use help to find out what to do (choose one)? o Yes o No 2. How many times for help? Why? 3. Does it search and view target force successfully? o Yes o No 4. Number of slips? What slips?	

	5. Number of errors? What errors?	
Search patterns which have the same specific force	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it search related patterns successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	
View each pattern from list (clicking the right key of mouse)	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it display each selected pattern successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	

Scenario 2:

Time Started: _____ Time Completed: _____

Total number of slips: _____

Total number of errors: _____

Main tasks:

Main tasks	Action	Notes
Searching target pattern	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it find the target pattern successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Displaying pattern content in current interface (by double clicking)	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it display the content of target pattern successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Displaying pattern content in new interface (by clicking the right key of mouse)	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it display the content of target pattern successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	

Creating relationship	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it create the relationship link between two patterns successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	

Scenario 3:

Time Started: _____ Time Completed: _____

Total number of slips: _____

Total number of errors: _____

Main tasks:

Main tasks	Action	Notes
Open the Shield.rtf file in appropriate field.	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it open the pattern file successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Transferring data by using context menu	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it transfer the selected content of the pattern successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Undo previous operation	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it undo previous operations successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	
Saving the pattern	1. Use help to find out what to do (choose one)? a. Yes b. No 2. How many times for help? Why? 3. Does it save the content of the pattern successfully? a. Yes b. No 4. Number of slips? What slips? 5. Number of errors? What errors?	

Appendix I: Task Scenarios for Evaluating the MUIP

Welcome to the MUIP Evaluation

For this evaluation, we are interested in the assessing the usability and functionality of the pattern management system MUIP. The system is created to aid researchers or user interface designers to manage UI pattern collections.

Before you go to complete scenarios, you first spend 10 minutes in exploring the MUIP:

1. Click "Register" button and enter your username into the popup window. Click "Commit" button to save your username.
2. Select your username from the Name list in the Login window.
3. Click "Login" to log on the system.
4. Explore the system.

NB: If you have any question regarding the system while exploring it, you can ask for help.

Now you start to complete the scenarios below:

The following pages contain five scenarios for you to complete which will help us assess the usability and functionality of the MUIP. When you are completing these scenarios, it is very important for us to know what is going on. The completion of the first three scenarios will be observed. Thus, as you complete each scenario of those three scenarios, please tell us what you are looking for, what you are thinking about, and what is confusing to you, etc?

Scenario 1: (Browsing patterns and forces)

There are some existing patterns in Mary Zajicek collections within the system. You know the patterns from this collection are created based on a set of forces. You want to see which patterns solve the same forces in the collection. Unfortunately, you cannot remember the concrete name of patterns. You login the system.

1. Enter the "Mary" in the collection searching field and author name field.
2. Click the "Search" button.
3. Look at the searching results which include ID, Version, Name, and Problem.
4. Select one pattern from the result list and click the "View".
5. Browse the detail of the selected pattern by viewing different tabs.
6. Select the "Problem" tab.
7. Select a force from the forces list and see the detail.
8. You have the opportunity to search a force from the system. Enter "F2" into the searching field and do the search.
9. Select the results and view the detail.
10. Now you are interested in a specific force of the pattern and want to see which other patterns solve the same force in this collection. You enter the label of the force into the force field within pattern finder section and do the search.
11. You can select and view each pattern from the searched results in another form (by clicking on the right key of mouse).

Scenario 2: (Creating relationship between patterns)

The Mary Zajicek collection contains "Error Recovery Loop" pattern and the "Context Sensitive Help Message" pattern. Both of them have the same version:3. Currently, you want to view the contents of them so that you can explore and create the possible relationship between them. So you login the system.

1. Enter the "Error Recovery Loop" pattern name into searching area and search the pattern.
2. Once you find the "Error Recovery Loop" pattern, you go to view the contents (by double

clicking)

3. After you read the details of the "Error Recovery Loop" pattern, you also want to view the contents of the "Context Sensitive Help Message" pattern. You can search for the "Context Sensitive Help Message" pattern based on Mary Zajicek collection name.
4. You have the opportunity to select and display the contents of "Context Sensitive Help Message" pattern in another form (by clicking the right key of mouse).
5. Look at the data from both patterns and do the analysis to identify that "Error Recovery Loop Pattern" pattern contains the "Context Sensitive Help Message" pattern.
6. In the interface containing "Error Recovery Loop Pattern" pattern, you select "Related Patterns&Log" tab to view the pattern link.
7. In this tab, you identify the "Context Sensitive Help Message" pattern by selecting the "Pattern ID", "Collection" and "Version number". You choose "Contains" relationship type and complete the relationship link by clicking "Add" button.

Scenario 3: (Importing a pattern manually)

You have copied a pattern named Shield from Welie's collection and saved as the Shield.rtf document in your computer. Within the "PLML" tab in the system, pattern written in rich text file will be displayed in the left side of this tab as well as the PLML will be shown in the right side of the tab. Now you want import the Shield pattern into the MUIP system.

1. View the tab "PLML".
2. Click the "Import" button in the "Pattern" sub-section and use the provided dialog to open and display the Shield.rtf file in this section from your computer.
3. Select information data of the pattern without heading.
4. Click the right key of mouse to show the Context Menu and select appropriate option in the Context Menu.
5. Check and confirm the operation already put the selected data into correct field.
6. If there are wrong operations, you can use the "Undo" to cancel previous operations.
7. Continue to transfer all data into appropriate fields.
8. Commit the pattern into the system.

Scenario 4: (Creating new version for a pattern)

You know the "Error Recovery Loop" only have the version 3. You want to create a new version for this pattern based on the version 3 so that you could modify the new version without changing the original one.

1. Search for the "Error Recovery Loop" in the version 3.
2. View the contents of the pattern.
3. Put a new version number "3.1" in the appropriate field.
4. Click the "New version" in the main menu and confirm to create the new version for this pattern.
5. Modify the new version of the pattern.
6. Save the change.
7. Browse two versions of the pattern and see the difference.

Scenario 5: (Creating a new collection)

You have browsed some patterns in the system. You want to create a new collection by using "Collection test" as name and collect some patterns with specific version number into a new collection for further use.

1. You can click the "Collection" in the main menu to view the collection form.
2. Enter "Collection test" as the collection name as well as other details regarding this collection.
3. Save the new collection information.
4. Select a target pattern from the system and collect into the new collection.
5. Continue to select and collect other patterns you want to put into the new collection.
6. Browse the patterns within the collection.

Appendix J: Questionnaire for Evaluating the MUIP

No.	Item	Strongly agree (1)	Agree (2)	Ok (3)	Disagree (4)	Strongly disagree (5)
1	Working with MUIP is easy.					
2	It takes too much time to learn how to use it.					
3	It provides enough guidance to perform tasks.					
4	The help documentation makes me confused.					
5	It is straightforward to get the task done.					
6	Error prevention messages are adequate.					
7	The response from the system is too slow.					
8	The interface of MUIP is logical and attractive.					
9	I would like to use MUIP to manage my own UI pattern collections.					
10	MUIP can import patterns written in RTF format.	Yes		No		
11	It is complex to convert other pattern forms into PLML.					
12	MUIP can export patterns written in PLML.	Yes		No		
13	MUIP can export patterns written in alternative formats.	Yes		No		
14	Editing the content of a pattern is convenient.					
15	It is useful to have multiple versions for a pattern.					
16	MUIP supports sufficient search functions to find target patterns from the system.					
17	A pattern in the system can be found in different ways.					
18	It takes many steps to create a new collection.					
19	Navigating patterns within a collection					

	is difficult.					
20	The way of manipulating collections is confusing.					
21	It is handy to browse existing forces.					
22	MUIP allows a user to reuse a force from the repository.	Yes		No		
23	It is obvious that users can create relationships between any two patterns.					
24	Naming new types of relationship between patterns is difficult.					
25	MUIP provides enough functions to assist me exploring relationship between patterns.					
26	MUIP provides enough functions to manipulate pattern collections.					
	Open-ended questions					
27	What features do you suggest adding to MUIP to further support managing pattern collections?					
28	Would you like to outline the advantages of the system?					
29	Would you like to outline disadvantages of the system?					
30	Would you like to write further comments regarding this system?					

References

Aiken, P., & Allen, D. (2004). XML for Data Management. San Francisco: Elsevier Inc.

Alexander, C. (1964). Notes on the Synthesis of Form. Cambridge: Harvard University Press.

Alexander, C. (1979). The Timeless Way of Building. New York: Oxford University Press.

Alexander, C., Ishikawa, S., Silverstein, M., Jacobson, M., Fiksdahl-King, I., & Angel, S. (1977). A Pattern Language: Towns, Building, Constructions. New York: Oxford University Press.

Alexander, C., Silverstein, M., Angel, S., Ishikawa, S., & Abrams, D. (1975). The Oregon Experiment. New York: Oxford University Press.

Appleton, B. (1997). Patterns and Software: Essential Concepts and Terminology. Retrieved 19th, December, 2004, from http://www.bradapp.net/

Beck, K., & Cunningham, W. (1987). Using Pattern Languages for Object-Oriented Programs. Paper Presented at the OOPSLA-87 Workshop on the Specification and Design for Object-Oriented Programming.

Bienhaus, D. (2004). PLMLx Doc. Retrieved 3rd, July, 2005, from http://www.cs.kent.ac.uk/people/staff/saf/patterns/diethelm/plmlx_doc/index.html

Bly, S. (1997). Field Work: Is It Product Work? Interactions. New York: Association for Computing Machinery.

Borchers, J. (1999). CHI Meets PLoP: An Interaction Patterns Workshop. Paper presented at the CHILiPLoP'99 Conference on Pattern Languages of Programming, Wickenburg.

Borchers, J. (2001). A Pattern Approach to Interaction Design. Chichester: John Wiley & Sons Ltd.

Brown, C.M.L. (1988). Human-Computer Interface Design Guidelines. Norwood: Albex Publishing Corporation.

Connolly, T.M., & Begg, C.E. (2004). Database Systems: A Practical Approach to Design, Implementation, and Management (Fourth Edition). Boston: Addison-Wesley Publishing Company.

Coplien, J.O. (1996). Software Patterns. New York: SIGS Books & Multimedia.

Coplien, J.O., & Schmidt, D.C. (1995). Pattern Languages of Program Design: Addison-Wesley Publishing Company.

Cunningham, W. (1994). About the Portland Form. Retrieved 28th, November, 2004, from http://c2.com/ppr/about/portland.html

Deng, J., Kemp, E., Todd, E.G. (2005). Managing UI Pattern Collections. Proceedings of the 6th ACM SIGCHI New Zealand Chapter's International Conference on Computer-Human Interaction: Making CHI Natural, Auckland, New Zealand.

DIAC02. (2002). Retrieved 12th, December, 2004, from http://diac.cpsr.org/cgi-bin/diac02

Dix, A., Finlay, J., Abowd, G.D., & Beale, R. (2004). Human-Computer Interaction (Third Edition): Pearson Education Limited.

DuBois, P. (2000). MySQL. Indianapolis: New Riders Publishing.

Duyne, D.K.V., Landay, J.A., & Hong, J.I. (2003). The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience. Boston: Addison-Wesley Publishing Company.

Erickson, T. (2000). Lingua Francas for Design: Sacred Places and Pattern Languages. Proceedings of the Conference on Designing Interactive Systems: Processes, Practices, Methods, and Techniques, New York, USA.

Fincher, S. (2003). CHI2003 Workshop Report Perspectives on HCI Patterns: Concepts and Tools (Introducing PLML). Interfaces, 56, 26-28: British HCI Group.

Fincher, S. (2004). PLML. Retrieved 12th, July, 2005, from http://www.cs.kent.ac.uk/people/staff/saf/patterns/concerns.html

Fincher, S., Finlay, J., Greene, S., & Molina, P.J. (2003). Perspectives on HCI Patterns: Concepts and Tools. Paper presented at the CHI2003 Perspectives on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

Fincher, S., & Windsor, P. (2000). Why Patterns Are Not Enough: Some Suggestions Concerning an Organising Principle for Patterns of UI Design. Paper presented at the CHI2000 Design Patterns Workshop, Hague, Netherlands.

Fowler, M., & Scott, K. (1997). UML Distilled Applying the Standard Object Modeling Language: Addison Wesley Longman, Inc.

Furtado, E., Furtado, V., Sousa, K.S., Vanderdonckt, J., & Limbourg, Q. (2004). KnowiXML: A Knowledge-Based System Generating Multiple Abstract User Interfaces in

USIXML. Proceedings of 3rd International Workshop on Task Models and Diagrams for User Interface Design TAMODIA'2004, Prague, Czech Republic.

Gabriel, R. (n.d.). Pattern Definition Thread. Retrieved 12th, December, 2004, from http://c2.com/cgi/wiki?PatternDefinionThread

Gaffar, A., Sinnig, D., Javahery, H., & Seffah, A. (2003). MOUDIL: A Comprehensive Framework for Disseminating and Sharing HCI Patterns. Paper presented at the CHI2003 Workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

Gaffar, A., Sinnig, D., Javahery, H., & Seffah, A. (2004). MOUDIL. Retrieved 28th, November, 2004, from http://hci.cs.concordia.ca/moudil

Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1995). Design Patterns: Elements of Reusable Object-Oriented Software: Addison-Wesley Publishing Company.

Goetz, J.P., & LeCompte, M.D. (1984). Ethnography and Qualitative Design in Educational Research. Orlando: Academic Press.

Graham, I. (2003a). A Pattern Language for Web Usability. London: Addison-Wesley Publishing Company.

Graham, I. (2003b). Wu Website. Retrieved 3rd, December, 2004, from http://www.wupatterns.com/

Granlund, Å., Lafrenière, D., & Carr, D.A. (2001). A Pattern-Supported Approach to the User Interface Design Process. Proceedings of 9th International Conference on Human-Computer Interaction, 2001, New Orleans, USA.

Greene, S.L., Matchen, P.M., Jones, L., Thomas, J.C., & Callery, M. (2003). Tool-Based Decision Support for Pattern Assisted Development. Paper presented at the CHI2003 Workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

Greene, S.L., Paul, M.M., & Jones, L. (2002). Tools for Pattern Use. Paper presented at the CHI2002: Patterns in Practice: A Workshop for UI Designers, Minneapolis, Minnesota, USA.

Griffiths, R.N., & Pemberton, L. (2001). Don't Write Guidelines Write Patterns! Retrieved 13rd, December, 2004, from http://www.it.bton.ac.uk/staff/lp22/guidelinesdraft.html

Henninger, S. (2001). An Organizational Learning Method for Applying Usability Guidelines and Patterns. Proceedings of the 8th IFIP International Conference on Engineering for Human-Computer Interaction, Toronto, Canada.

Henninger, S., Keshk, M., & Kinworthy, R. (2003). Capturing and Disseminating Usability Patterns with Semantic Web Technology. Paper presented at the CHI2003 Workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

International, E.ECMA. (n.d.). Retrieved 10th, June, 2005, from http://www.ecma-international.org/

Jacko, J.A., & Sears, A. (2003). The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies, and Emerging Applications. Mahwah, New Jersey: Lawrence Erlbaum Associates.

Jacobson, I., Ericsson, M., & Jacobson, A. (1995). The Object Advantage: Business Process Reengineering with Object Technology. Wokingham: Addison-Wesley.

Kirakowski, J. (1994). The Use of Questionnaire Methods for Usability Assessment. Retrieved 10th, December, 2004, from http://www.ucc.ie/hfrg/questionnaires/sumi/sumipapp.html

Laakso, S.A. (2003). User Interface Design Patterns. Retrieved 28th, November, 2004, from http://www.cs.helsinki.fi/u/salaakso/patterns/index.html

Lewandowski, S.M. (1998). Frameworks for Component-Based Client/Server Computing. ACM Computing Surveys, 30(1), 3-27.

Lhotka, R. (2003). Expert One-on-One Visual Basic.NET Business Objects: Apress.

Lin, J., & Landay, J. (2003). Damask: A Tool for Early-Stage Design and Prototyping of Cross-Device User Interfaces. Paper presented at the CHI2003 Workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

Mahemoff, M.J., & Johnston, L.J. (1998a). Pattern Language for Usability: An Investigation of Alternative Approaches. Proceedings of Asia-Pacific Conference on Human Computer Interaction APCHI'98, Shonan Village, Japan.

Mahemoff, M.J., & Johnston, L.J. (1998b). Principles for a Usability-Oriented Pattern Language. Proceedings of the Australian Computer Human Interaction Conference OZCHI'98, Adelaide, Australia.

Malone, T.B., Kirkptrick, M., & Heasly, C.C. (1984). Human-Computer Interface Effectiveness Evaluation. Proceedings of the First U.S.A-Japan conference on Human-Computer Interaction, Honolulu, Hawaii.

Martin, D., Rodden, T., Sommerville, I., Rouncefield, M., & Hughes, J. (2004). Patterns of cooperative interaction. Retrieved 28th, November, 2004, from http://www.comp.lancs.ac.uk/computing/research/cseg/projects/pointer/patterns.html

Mayhew, D.J. (1997). Principles and Guidelines in Software User Interface Design (First Edition): Pearson Education Limited.

Merrick, R.A. (2001). AUIML: An XML Vocabulary for Describing User Interfaces. [Device Independent User Interfaces in XML.]. Retrieved June 10th, 2005, from http://www.belchi.be/download/merrick.pdf

Microsoft.ECMA and ISO/IEC C# and Common Language Infrastructure Standards. (n.d.). Retrieved June 10th, 2005, from http://msdn.microsoft.com/net/ecma/

Mok, H.N. (2003). From Java to C#: a developer's guide. New York: Addison-Wesley Publishing Company.

Montero, F., López-Jaquero, V., Vanderdonckt, J., González, P., & Lozano, M. (2005). Solving the Mapping Problem in User Interface Design by Seamless Integration in IDEALXML. Paper presented at the DSVIS'05 - 12th International Workshop on Design, Specification and Verification of Interactive Systems, Newcastle upon Tyne, England.

Mullet, K. (2002). Structuring Pattern Languages to Facilitate Design. Paper presented at the Patterns Workshop, CHI2002, Philidelphia.

Molina, P.J., & Hernandez, J. (2003). Just-UI: Using Patterns as Concepts for IU Specification and Code Generation. Paper presented at the CHI2003 Workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

Mozilla. (n.d.). Retrieved June 10th, 2005, from http://www.mozilla.org/projects/xul/

Mullet, K. (2002). Structuring Pattern Languages to Facilitate Design. Paper presented at the Patterns Workshop, CHI2002, Philidelphia.

Myers, B.A. (1995). User Interface Software Tools. ACM Transactions on Computer-Human Interaction, 2(1), 64-103.

Netscape. (n.d.). Retrieved June 10th, 2005, from http://developer.netscape.com/library/documentation/communicator/jsguide4/index.htm

Noble, J. (1998). Classifying Relationships between Object-Oriented Design Patterns. Paper presented at the Australian Software Engineering Conference, Adelaide.

Norman, D.A. (1988). The Psychology of Everyday Things. New York: Basic Books.

Norman, D.A., & Drapher, S.W. (1986). User-Centered System Design: New Perspectives on Human-Computer Interaction. Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Paternò, F., Mancini, C., & Meniconi, S. (1997). ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. Proceedings of the IFIP TC13 International Conference on Human-Computer Interaction, Sydney.

Pattern Definition Thread. (n.d.). Retrieved 12th, December, 2004, from http://c2.com/cgi/wiki?PatternDefinition

Pemberton, L. (2000). The Promise of Pattern Languages for Interaction Design. Retrieved 19th, December, 2004, from http://www.it.bton.ac.uk/staff/lp22/HF2000.html

Preece, J., Rogers, Y., & Sharp, H. (2002). Interaction Design: Beyond Human-Computer Interaction: John Wiley & Sons, Inc.

Puerta, A., & Eisenstein, J. (2001). XIML: A Universal Language for User Interfaces. Retrieved 19th, December, 2004, from www.ximl.org/documents/XIMLBasicPaperES.pdf

Puerta, A., & Eisenstein, J. (2002). XIML: A Common Representation for Interaction Data. Paper presented at the IUI'02, San Francisco, California, USA.

Robson, C. (1993). Real World Research. Blackwell: Oxford press.

Rowley, D.E., & Rhoades, D.G. (1992). The Cognitive Jogthrough: A Fast-Paced User Interface Evaluation Procedure. Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, Monterey, California, United States.

Salingaros, N.A. (2000). The Structure of Pattern Languages: Cambridge University Press.

Schuemmer, T. (2003a). Evolving a Groupware Pattern Language. Paper presented at the ECSCW2003 Workshop "From Good Practices to Patterns", Helsinki, Finland.

Schuemmer, T. (2003b). Seeking for Structure in a Groupware Pattern Language. Paper presented at the CHI 2003 Workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

Simarro, F.M. (2005). IDEALXML: Interface Development Environment for Applications Specified in usiXML. Retrieved 19th, June, 2005, from http://www.usixml.org/index.php?view=page&idpage=34&screen_size=

Sinha, A. (1992). Client-Server Computing. Communications of the ACM, 35(7), 77-98.

Sinnig, D., Forbrig, P., & Seffah, A. (2003). Patterns in Model-Based Development. Paper presented at the 2nd Workshop on Software and Usability Cross-Pollination: The Role of Usability Patterns, Zurich, Switzerland.

Skogseid, I., & Spring, M. (1995). Patterns for Human-Computer Interaction Studies of Principle Aggregation and Pattern Naming. Retrieved 5th, December, 2004, from http://www.sis.pitt.edu/~spring/patterns/patterns.html

Smith, S.L., & Mosier, J.N. (1986). Guidelines for Designing User Interface Software (No. Technical Report MTR-010090, ESD-TR-86-278). Bedford, Massachusetts, USA: The Mitre Corporation.

Stone, D., Jarrett, C., Woodroffe, M., & Minocha, S. (2005). User Interface Design and Evaluation. San Francisco: Morgan Kaufmann.

SUMI. (n.d.). Retrieved 8th, June, 2005, from http://www.ucc.ie/hfrg/questionnaires/sumi/

Sutcliffe, A.G. (1995). Human-Computer Interface Design. Houndmills: Macmillan Press Ltd.

Tidwell, J. (1999). Common Ground: A Pattern Language for Human-Computer Interface Design. Retrieved 28th, November, 2004, from http://www.mit.edu/~jtidwell/interaction_patterns.html

Tidwell, J. (2002). UI Patterns and Techniques. Retrieved 28th, November, 2004, from http://time-tripper.com/uipatterns/index.php

Todd, E., Kemp, E., & Philips, C. (2004). What Makes a Good User Interface Pattern Language? Paper presented at the 5th Australasian User Interface Conference, Dunedin.

UIML.Org. (n.d.). Retrieved 19th, December, 2004, from http://www.uiml.org/index.php

UsiXML.Org. (n.d.). Retrieved 19th, December, 2004, from http://www.usixml.org/index.php?view=news

W3C.CSS. (n.d.). Retrieved 19th, December, 2004, from http://www.w3.org/Style/CSS/

W3C.DOM. (n.d.). Retrieved 19th, December, 2004, from http://www.w3.org/DOM/

W3C.HTML. (n.d.). Retrieved 19th, December, 2004, from http://www.w3.org/TR/REC-html40/

W3C.XForm. (n.d.). Retrieved 19th, December, 2004, from http://www.w3.org/MarkUp/Forms/

W3C.XML. (n.d.). Retrieved 19th, December, 2004, from http://www.w3.org/XML/

Welie, M.v. (2001). Task-Based User Interface Design. Unpublished PhD Thesis, Vrije Universiteit.

Welie, M.v. (2003). Generative Pattern Languages for Interaction Design. Paper presented at the CHI2003 Workshop on HCI Patterns: Concepts and Tools, Fort Lauderdale, Florida.

Welie, M.v. (2004a). Interaction Design Patterns, Retrieved 28th, November, 2004, from http://www.welie.com/patterns/index.html

Welie, M.v. (2004b). Patterns in Interaction Design. Retrieved 28th, November, 2004, from http://www.welie.com/index.html

Welie, M.v., Mullet, K., & McInerney, P. (2002). Patterns in Practice: A Workshop for UI Designers. Paper presented at the CHI2002 Extended Abstracts on Human Factors in Computing Systems, Minneapolis, Minnesota, USA.

Welie, M.v., & Trætteberg, H. (2000). Interaction Patterns in User Interfaces. Paper presented at the Pattern Languages of Programs Conference, Allerton Park Monticello, Illinois, USA.

Welie, M.v., & Veer, G.C.v.d. (2003). Pattern Languages in Interaction Design: Structure and Organization. Proceedings of Interact '03, Zurich, Switzerland.

Welie, M.v., Veer, G.C.v.d., & Eliens, A. (2000). Patterns as Tools for User Interface Design. Paper presented at the International Workshop on Tools for Working with Guidelines, Biarritz, France.

XAML. (n.d.). Retrieved 30th, December, 2004, from http://longhorn.msdn.microsoft.com/lhsdk/core/overviews/about%20xaml.aspx

XForms. (n.d.). Retrieved 28th, December, 2004, from http://www.w3.org/MarkUp/Forms/

XIML.Org. (n.d.). Retrieved 28th, December, 2004, from http://www.XIML.org/

Zajicek, M. (2004). Eight Patterns for Speech Interaction for Older Adults. Paper presented at the CHI2004 Conference on Human Factors in Computing Systems, Vienna, Austria.

Zimmermann, G., Vanderheiden, G., & Gilman, A. (2002). Universal Remote Console Prototyping of an Emerging XML Based Alternate User Interface Access Standard. Paper presented at the 11th International World Wide Web Conference, Honolulu, Hawaii.