

American University in Cairo

## AUC Knowledge Fountain

---

Archived Theses and Dissertations

---

5-1-2005

### PLITS

Dina Salah El Din Nasr

*The American University in Cairo AUC*

Follow this and additional works at: [https://fount.aucegypt.edu/retro\\_etds](https://fount.aucegypt.edu/retro_etds)



Part of the [Programming Languages and Compilers Commons](#)

---

#### Recommended Citation

##### APA Citation

Nasr, D. (2005). *PLITS* [Thesis, the American University in Cairo]. AUC Knowledge Fountain.

[https://fount.aucegypt.edu/retro\\_etds/1935](https://fount.aucegypt.edu/retro_etds/1935)

##### MLA Citation

Nasr, Dina Salah El Din. *PLITS*. 2005. American University in Cairo, Thesis. *AUC Knowledge Fountain*.

[https://fount.aucegypt.edu/retro\\_etds/1935](https://fount.aucegypt.edu/retro_etds/1935)

This Thesis is brought to you for free and open access by AUC Knowledge Fountain. It has been accepted for inclusion in Archived Theses and Dissertations by an authorized administrator of AUC Knowledge Fountain. For more information, please contact [fountadmin@aucegypt.edu](mailto:fountadmin@aucegypt.edu).

**PLITS: A PATTERN  
LANGUAGE FOR  
INTELLIGENT  
TUTORING SYSTEMS**

**DINA SALAH  
EL DIN NASR**

**2005**

2005/62

The American University in Cairo  
School of Science and Engineering



# **PLITS: A Pattern Language for Intelligent Tutoring Systems**

A Thesis Submitted to

The Department of Computer Science

in partial fulfillment of the requirements for the

Degree of Master of Science

By

**Dina Salah El Din Nasr**

Under the Supervision of

**Dr. Amir Zeid**

May 2005

# PLITS: A Pattern Language for Intelligent Tutoring Systems

A Thesis Submitted by

Dina Salah El Din Nasr

To The Department of Computer Science

May 2005

In partial fulfillment of the requirements for

The degree of Master of Science

Has been approved by

Dr. [Redacted]

*for* Thesis Committee Chair / Advisor

Affiliation AUC

Dr. .... [Redacted]

Thesis Committee Reader / Examiner

Affiliation AUC

Dr. [Redacted]  
Thesis Committee Reader / Examiner

Affiliation AUC

Dr. [Redacted]  
Thesis Committee Reader / External Examiner

Affiliation Faculty of Computer & Information Sciences,  
ams Unit.

[Redacted]  
Department Chair

Date

June 4, 05

[Redacted]  
Dean

Date

June 6, 2005



The American University in Cairo

LIBRARY

Thesis

Declaration to be signed by the Author

Name: .....Dina Salah El Din Nasr.....

Title of the Thesis: .....PLITS: A pattern language for  
.....Intelligent Tutoring Systems.....

Department:.....Computer..... Year: .....2005..... (Library No.28534000)30811  
Science

Please Sign and date ONE of the following paragraphs:

1. The Thesis may be consulted in the Library and Photocopied.

Signed:..... Date:.....

OR

2. The Thesis may be consulted in the Library, but may not be photocopied.  
The restriction on photocopying will cease two years after the date below,  
unless I apply for, and am granted its renewal. \*

Signed:..... Date:.....

OR

3. The Thesis may neither be consulted nor photocopied without written  
permission from me or from the appropriate Head of Department if I cannot  
be contacted. This restriction will cease three years after the date below,  
unless I apply for, and am granted its renewal. \*

Signed:.....Dina Salah El Din Nasr..... Date:.....24/5/2005.....

\* Application for renewal of restrictions will be considered by the Librarian, the appropriate Head of Department and the Chairman of the Academic Board or his nominee.

## Dedication

If the family were a container, it would be a nest, an enduring nest, loosely woven, expansive, and open. If the family were a fruit, it would be an orange, a circle of sections, held together but separable—each segment distinct. If the family were a boat, it would be a canoe that makes no progress unless everyone paddles.

"Letty Cottin Pogrebin "

Call it a clan, call it a network, call it a tribe, call it a family. Whatever you call it, whoever you are, you need one.

"Jane Howard "

To my lovely parents, my dear brother and my friend Nisreen.

## Acknowledgments

A teacher affects eternity; he can never tell where his influence stops.

"Henry Brooks Adams"

It has been a pleasure working with Dr.Amir Zeid in producing my thesis work. Dr.Zeid expressed excellent academic, teaching and technical skills. I would really like to acknowledge Dr.Zeid for all his efforts in helping me to accomplish this thesis work successfully.

I had the pleasure to have Dr.Hoda Hosny and Dr.Sherif El Kassas as thesis committee members. I would like to express my deepest gratitude for them for all the time and effort they have provided me.

I would like to dedicate special thanks to Mr.Abbas Tonsi, Director of the Arabic Writing Program and Director of Developing Material and Computer Aided Language Learning Unit at the Arabic Language Institute in the American University in Cairo for providing me with all the necessary resources to test my experiments.

I had the honor to have Dr.Osman Mohamed Ibrahim, Visiting Professor Faculty of Computers and Information Ain Shams University, as an external examiner. I would like to express my deepest gratitude for him for all the valuable academic and technical comments he gave me concerning my research work and directions for future research. I would really like to acknowledge Dr.Osman for all his efforts in helping me to accomplish this thesis work successfully.

Finally, I would like to express my deep gratitude to my parents and my brother who strongly helped me to finish this thesis work successfully.

The American University in Cairo

Abstract

**PLITS: A Pattern Language for Intelligent Tutoring Systems**

By Dina Salah El Din Nasr

Supervised By

Dr.Amir Zeid

The design and implementation of Intelligent Tutoring Systems (ITS) is a very complex task, as it involves a variety of organizational, administrative, instructional and technological components. In addition, there are no well established methodologies or development tools for ITS implementation. Therefore systematic, disciplined approaches must be devised in order to leverage the complexity of ITS implementation and achieve overall product quality within specific time and budget limits. Moreover, the goal of patterns within software community is to provide software developers with solutions to recurring software problems.

In this research we tried to mine patterns by reverse-engineering the ITS systems that embed good design in order to make that design explicit, and be able to communicate it to other designers, so that it becomes common practice. As a result, good design experience is codified and a more systematic development process for these systems is being provided. These patterns were semantically organized and categorized to form the basic core of a pattern language for Intelligent Tutoring Systems. This research work aims to take a few steps in this direction by proposing a pattern language for intelligent tutoring systems. Designers of new or existing ITS, especially in the field of Intelligent Language Tutoring Systems (ILTS), can thus take advantage of previous design expertise and save precious time and resources.



These patterns were used in designing the **Arabic Tutor** which is an Intelligent Language Tutoring System over the World Wide Web for teaching a subset of the Arabic Language that combines the flexibility and intelligence of Intelligent Tutoring Systems with the availability of the World Wide Web applications.

**Table of Contents**

Chapter 1: Introduction	1
Overview	1
1.2 Research Motivation	3
1.3 The Addressed Problem	5
1.4 Outcomes and Contributions	7
1.5 The Solution Approach	7
1.5.1 Phase One: Pattern Discovery	7
1.5.2 Phase Two: Formulating PLITS	8
1.5.3 Phase Three: Testing	8
1.5.3.1 Testing PLITS (By Implementing the Arabic Tutor)	8
1.5.3.2 Testing the Arabic Tutor	8
1.6 Thesis Organization	9
Chapter 2: Research Background	10
2.1 Reusability	10
2.2 Design Patterns	10
2.2.1 What is a pattern?	10
2.2.2 Features of Good Patterns	11
2.2.3 Anti-Patterns	12
2.2.4 Pattern Classification	12
2.2.5 Elements of a Pattern	13
2.2.6 Design Patterns and Frameworks	15
2.2.7 Pattern Languages	16
2.2.7.1 Pattern Language General Model	17
2.2.7.2 A Pattern Language Example	18

2.2.8 Pattern Catalogs and Systems	20
2.9 What is an Intelligent Tutoring System?	21
2.10 Intelligent Tutoring Systems versus Traditional Tutoring Systems	22
2.11 Components of Intelligent Tutoring Systems	22
2.12 History of Intelligent Tutoring Systems	23
2.13 Architecture of Intelligent Tutoring Systems	24
2.13.1 Architecture of Standalone Intelligent Tutoring Systems	24
2.13.2 Architectures of Intelligent Tutoring Systems on the World Wide Web	25
2.13.2.1 Centralized Architecture	25
2.13.2.2 Replicated Architecture	26
2.13.2.3 Distributed Architecture	26
2.13.2.4 Comparison Between the Different Web Based Architectures	27
2.13.3 Web based Intelligent Tutoring Systems versus Standalone Intelligent Tutoring Systems	28
Chapter 3: Related Work	29
3.1 Standalone Intelligent Language Tutoring Systems	30
3.2 Web Based Intelligent Language Tutoring Systems	32
3.3 Design and Analysis Patterns in ITS	33
3.3.1 KnowledgeModel View Pattern	33
3.3.2 Composite Pattern	34
3.3.3 Builder Pattern	36
3.4 A Pattern Language for Architectures of Intelligent Tutors	38
3.4.1 Inserted Layer	38
3.4.2 Top	39
3.4.3 Cascade	39
3.4.4 T-join	40

3.4.5 Cross	40
Chapter 4: The Proposed Pattern Language for Intelligent Tutoring Systems (PLITS)	41
4.1 PLITS Pattern Categories	42
4.2 PLITS Pattern Template	43
4.2.1 GOF Pattern Template	44
4.2.2 Alexandrian Pattern Template	45
4.3 Building an Intelligent Tutoring System	46
4.3.1 Phase One: Building the Domain Module	47
4.3.1.1 Pattern 1: Early Bird	47
4.3.1.2 Pattern 2: The Whole Part Pattern	49
4.3.1.3 Pattern 3: Singleton	53
4.3.1.4 Pattern 4: New Flyweight	56
4.3.1.5 Pattern 5: Builder	61
4.3.2 Phase Two: Building the Student Model	65
4.3.2.1 Pattern 6: Registration-Authentication-Access Control	65
4.3.2.2 Pattern 7: User Model Definition	66
4.3.2.3 Pattern 8: User Goals	69
4.3.2.4 Pattern 9: New Student Model Initialization	71
4.3.2.5 Pattern 10: New Student Model Maintenance	74
4.3.2.6 Pattern 11: New Adapter Also Known as Wrapper	77
4.3.3 Phase Three: Building the Tutor Module	82
4.3.3.1 Whole Part Pattern	82
4.3.3.2 Pattern 12: New Strategy Also Known as Policy	83
4.3.3.3 Pattern 13: Template Method	87

4.3.3.4 Pattern 14: Master Slave	90
4.3.3.5 Singleton Pattern	93
4.3.3.6 Pattern 15: Observer Also Known as Dependents, Publish Subscribe	93
4.3.4 Phase Four: Building the Graphical User Interface Module	98
4.3.4.1 Singleton Pattern	98
4.3.4.2 Pattern 16: Course Creation and Customization	99
4.3.4.3 Pattern 17: Wizard	102
4.3.4.4 Pattern 18: Shield	105
4.3.4.5 Pattern 19: Warning	107
4.4 PLITS: A Pattern language for Intelligent Tutoring Systems	108
Chapter 5: The Arabic Tutor	113
5.1 The Arabic Tutor Components	113
5.1.1 Pedagogical Module (Tutor Module)	113
5.1.2 Expert Module	113
5.1.3 Domain Module	114
5.1.4 Student Model	114
5.1.5 GUI Module	115
5.2 The Arabic Tutor Architecture	115
5.3 The Arabic Tutor Features	116
5.4 Building the Arabic Tutor Using PLITS	117
5.4.1 Early Bird Pattern in the Arabic Tutor	118
5.4.2 Whole Part Pattern in the Arabic Tutor	119
5.4.3 Singleton Pattern in the Arabic Tutor	121
5.4.4 New Flyweight Pattern in the Arabic Tutor	121
5.4.5 Builder Pattern in the Arabic Tutor	123
5.4.6 Registration-Authentication-Access Control Pattern in the Arabic Tutor	125

5.4.7 User Model Definition Pattern in the Arabic Tutor	126
5.4.8 User Goals Pattern in the Arabic Tutor	127
5.4.9 Shield Pattern in the Arabic Tutor	127
5.4.10 New Student Model Initialization Pattern in the Arabic Tutor	129
5.4.11 Template Method Pattern in the Arabic Tutor	131
5.4.12 Master Slave Pattern in the Arabic Tutor	139
5.4.13 New Adapter Pattern in the Arabic Tutor	141
5.4.14 New Student Maintenance Pattern in the Arabic Tutor	143
5.4.15 New Strategy Pattern in the Arabic Tutor	146
5.4.16 Wizard Pattern in the Arabic Tutor	148
5.4.17 Course Creation and Customization Pattern in the Arabic Tutor	165
5.4.18 Warning Pattern in the Arabic Tutor	167
5.4.19 Observer Pattern in the Arabic Tutor	178
5.5 The Arabic Tutor Class Diagram	183
5.6 Results of Arabic Tutor Evaluation	185
Chapter 6: Summary and Conclusion	186
6.1 Achievements and Contributions	186
6.1.1 Patterns Discovery	186
6.1.2 Formulating PLITS	187
6.1.3 Implementing the Arabic Tutor	187
6.2 Directions for Future Research	187
Bibliography	190
Appendix A : The Arabic Tutor Development Environment	194
Appendix B: Arabic Alphabet Unicode Mapping	195
Appendix C: Arabic Lessons, Examples and exercises	196
Appendix D: Sample Source Code	203

Appendix E: How to Read a UML Class Diagram	235
Appendix F: Detailed Class Diagrams	236
Appendix G: Sample Expert System Source Code using Jess for Future Work	249

## List of Tables

Table (1): ITS and Traditional TS Comparison	22
Table (2): Standalone Intelligent Language Tutoring Systems	30
Table (3): Web Based Intelligent Language Tutoring Systems	32
Table (4): ITSs That Were Searched for Patterns	41
Table (5): Whole Class Responsibility and Collaborators	50
Table (6): Part Class Responsibility and Collaborators	51
Table (7): Master Class Responsibility and Collaborators	92
Table (8): Slave Class Responsibility and Collaborators	92
Table (9): Mapping Between the Functional Requirements and the Corresponding ITS Patterns	110
Table (10): Implementation of Question Type in the Template Method	132
Table (11): Difficulty Level Criteria for Arabic Tutor Exercises	133
Table (12): Students' Score Computation Method in Entry Exam in the Arabic Tutor	141
Table (13): Students' Stereotypes	143
Table (14): Teaching Material According to Student Level.	145
Table (15): The Per Topic Exam Configurable Parameters in the Arabic Tutor	151
Table (16): Per Topic Exam Scores Computation Method in the Arabic Tutor	153
Table (17): The Arabic Tutor Configurable Parameters	158
Table (18): Teaching Material According to Student Level	161



## List of Figures

Figure (2-1): A Partial Pattern Language for Web Design (Centered Around Shopping)	19
Figure (2-2): Need for Personalized Instruction	21
Figure (2-3): The Student Model	23
Figure (2-4): Standalone ITS Architecture	25
Figure (2-5): Centralized Web ITS Architecture	26
Figure (2-6): Replicated Web ITS Architecture	26
Figure (2-7): Distributed Web ITS Architecture	26
Figure (3-1): NLP Modules	31
Figure (3-2): CAPIT Architecture	31
Figure (3-3) : The German Tutor	33
Figure (3-4): KnowledgeModel View Pattern	34
Figure (3-5): Structure of the Composite Pattern	35
Figure (3-6): Using the Builder Pattern in Designing Explanation Generator	37
Figure (3-7): General Structure of the Builder Pattern	37
Figure (3-8) a) The Inserted Layer Pattern b) The Top Pattern c) The Cascade Pattern d) The T-join Pattern e) The Cross Pattern	39
Figure (4-1): Whole Part Pattern Structure	50
Figure (4-2): Singleton Pattern Structure	54
Figure (4-3): New Flyweight Pattern Structure	58
Figure (4-4): Flyweight Pattern Structure	58
Figure (4-5): Builder Pattern Structure	62
Figure (4-6): Using Builder Pattern in Implementing Explanation Generator in Get-Bits	62
Figure (4-7): New Adapter Pattern Structure	80
Figure (4-8): Adapter Pattern Structure	80
Figure (4-9): New Strategy Pattern Structure	84

Figure (4-10): Strategy Pattern Structure	84
Figure (4-11): Template Method Pattern Structure	89
Figure (4-12): Observer Pattern Structure	95
Figure (4-13): CAPIT Main User Interface	104
Figure (4-14): Dictation Exercise in the German Tutor	104
Figure (4-15): Which Word is Different in the Greek Tutor	105
Figure (4-16): PLITS: Pattern Language Approach	112
Figure (5-1): The Arabic Tutor System Architecture	113
Figure (5-2): Arabic Tutor Centralized Web Architecture	115
Figure (5-3): Whole Part Pattern Structure in the Arabic Tutor	120
Figure (5-4): Singleton Pattern in the TeachingMaterial Class in the Arabic Tutor	121
Figure (5-5): New Flyweight Pattern Structure in the TeachingMaterial Class in the Arabic Tutor	122
Figure (5-6): Builder Pattern Structure in the Arabic Tutor	124
Figure (5-7): Student Learning Agenda in the Arabic Tutor	125
Figure (5-8): Student Signup Form in the Arabic Tutor	126
Figure (5-9): Student Aborted Attempt to Start Learning Without Determining his Long Term Goals	128
Figure (5-10): Whole Part Pattern Structure in the Arabic Tutor (Exam-StudentExercises)	130
Figure (5-11): Template Method Pattern Structure in Arabic Tutor	133
Figure (5-12): Singleton Pattern in the UserInterfaceBundle Class in the Arabic Tutor	134
Figure (5-13): New Flyweight Pattern Structure in the ArabicAlphabeticBundle Class in the Arabic Tutor	137
Figure (5-14): Singleton Pattern in the ArabicAlphabeticBundle Class in the Arabic Tutor	138
Figure (5-15): Entry Exam Sample Exercises	138
Figure (5-16): Shield Pattern in the Arabic Tutor –Protecting Student from Unanswering a	139

Question	
Figure (5-17): Entry Exam Score Sheet	139
Figure (5-18): Master Slave Pattern Structure in the Arabic Tutor	140
Figure (5-19): New Adapter Pattern Structure in the Arabic Tutor (Exercise-StudentExcercise)	142
Figure (5-20): New Adapter Pattern Structure in the Arabic Tutor (Topic-StudentTopic)	144
Figure (5-21): New Strategy Pattern Structure in the Arabic Tutor	147
Figure (5-22):Topic1 Explanatory Text	148
Figure (5-23): First Learn Item Explanatory Text and examples for Topic1	149
Figure (5-24): Second Learn Item Explanatory Text and Examples for Topic1	150
Figure (5-25): Third Learn Item Explanatory Text and Examples for Topic1	150
Figure (5-26): Per Topic Exam Sample Questions	151
Figure (5-27): Shield Pattern in the Arabic Tutor – Protecting the Student from Semi Answering a Question	152
Figure (5-28): Overall Topic Score Sheet per Student	152
Figure (5-29):Topic2 Explanation Text	155
Figure (5-30): First Learn Item Explanation Text and Examples for Topic2	155
Figure (5-31): Second Learn Item Explanation Text and Examples for Topic2	156
Figure (5-32): Third Learn Item Explanation Text and Examples for Topic2	156
Figure (5-33): Sample per Topic Exam Questions for Topic2	157
Figure (5-34): Wrong Student Answer on an MCQ Question	158
Figure (5-35): Tailored Feedback	159
Figure (5-36): Wrong Student Answer on a True/False Question	160
Figure (5-37): Tailored Feedback for True/False Question	160
Figure (5-38): Overall Topic Score Sheet Per Student	161
Figure (5-39): Final Topic Score Sheet Per Student	162

Figure (5-40): Overall Topics Score Sheet Per Student	163
Figure (5-41): Teacher Login Screen	165
Figure (5-42): Topics Screen	165
Figure (5-43): Add Topic Screen	166
Figure (5-44): Delete Topic Confirmation Screen	166
Figure (5-45): Edit Topic Screen	168
Figure (5-46): Add Learn Item Screen	168
Figure (5-47): Delete Learn Item Confirmation Screen	169
Figure (5-48): Edit Learn Item Screen	170
Figure (5-49): Add Example Screen	170
Figure (5-50): Delete Example Confirmation Screen	171
Figure (5-51): Edit Example Screen	171
Figure (5-52): Delete Exercise Screen	172
Figure (5-53): Add Multiple Choice Question Screen	173
Figure (5-54): Yes/No Question Screen	174
Figure (5-55): Enter verb Question Screen	175
Figure (5-56): Singleton Pattern in the CollectiveStudentInfo Class in the Arabic Tutor	178
Figure (5-57): Observer Pattern Structure in the Arabic Tutor	179
Figure (5-58): Main Reports Screen	180
Figure (5-59): Collective Topic Progress Report	181
Figure (5-60): Student Progress in All Topics Report	183
Figure (5-61): The Arabic Tutor Class Diagram	184
Figure (F-1): Whole Part Detailed Class Diagram. TeachingMaterial-Topic-LearnItem-Example-Exercise	237
Figure (F-2): Whole Part Detailed Class Diagram. Exam-StudentExcercise	238
Figure (F-3): Singleton Pattern. TeachingMaterial	239

Figure (F-4): Singleton Pattern. ArabicAlphabeticBundle	239
Figure (F-5): Singleton Pattern. CollectiveStudentInfo	240
Figure (F-6): Singleton Pattern. UserInterfaceBundle	240
Figure (F-7): New Flyweight Pattern Detailed Class Diagram. ArabicAlphabeticBundle-Letter-Character	241
Figure (F-8): New Flyweight Pattern Detailed Class Diagram. TeachingMaterial-Topic-LearnItem-Exercise-Example	242
Figure (F-9): Builder Pattern Detailed Class Diagram	243
Figure (F-10): New Adapter Pattern Detailed Class Diagram. Exercise-StudentExcercise	244
Figure (F-11): New Adapter Pattern Detailed Class Diagram. Topic-StudentTopic	245
Figure (F-12): New Strategy Pattern Detailed Class Diagram	246
Figure (F-13): Template Method Pattern Detailed Class Diagram	247
Figure (F-14): Master Slave Pattern Detailed Class Diagram	248
Figure (F-15): Observer Pattern Detailed Class Diagram	249

**List of Abbreviations**

1	PLITS	A Pattern Language For Intelligent Tutoring Systems
2	ITS	Intelligent Tutoring Systems
3	ICALL	Intelligent Computer Aided Language Learning
4	ILTS	Intelligent Language Tutoring System
5	WWW	World Wide Web
6	TS	Tutoring Systems
7	CBT	Computer Based Training
8	HTML	Hyper Text Markup Language
9	CGI	Common Gateway Interface
10	TCP	Transmission Control Protocol
11	NLP	Natural Language Processing
12	CAPIT	Capitalization and Punctuation Intelligent Tutor
13	PLAIT	A Pattern Language for Architectures of Intelligent Tutors
14	Web PVT	Web Passive Voice Tutor
15	SQLT	Structured Query Language Tutor
16	SQL	Structured Query Language
17	CAD	Computer Aided Design
18	MCQ	Multiple Choice Questions
19	ALITS	Arabic Language Intelligent Tutoring Systems
20	POSA	Pattern Oriented Software Architecture
21	GOF	Gang of Four
22	GUI	Graphical User Interface

## Chapter 1

# Introduction

### 1.1 Overview

In software engineering, patterns are attempts to describe successful solutions to common software problems. Software patterns reflect common conceptual structures of these solutions, and can be applied over and over again when analyzing, designing, and producing applications in a particular context. Patterns help designers capture the core structure of the systems they build and provide a disciplined format for sharing ideas that too often remained locked in the heads of a few people [1].

So aside from recurrence, a pattern must describe *how* the solution balances or resolves its forces, and *why* this is a "good" resolution of forces. We need both of these to convince us that the pattern is neither a speculation (pure theory) nor is the pattern blindly following others (rote practice). We want to show that the practice is more than just "theory", and that the theory really has been practiced. We might even say that:

A pattern is where theory and practice meet to reinforce and complement one another, by showing that the structure it describes is useful, useable, and has been used [2].

A pattern must be *useful* because this shows how having the pattern in our minds may be transformed into an instance of the pattern in the real world, as something that adds value to our lives as developers and practitioners. A pattern must also be *useable* because this shows how a pattern described in literary form may be transformed into a pattern that we have in our minds. And a pattern must be *used* because this is how patterns that exist in the real world first became documented as patterns in literary form. This yields a continuously repeating cycle from pattern writers, to pattern readers, to pattern users: writers documenting patterns in literary

form make them usable to pattern readers, who can then remember them in their minds, which makes them useful to practitioners and developers, who can use them in the real world, and enhance the user's quality of life [2].

A pattern Language is a collection of patterns that form a vocabulary for understanding and communicating ideas. If a pattern is a recurring solution to a problem in a context given by some forces, then a pattern language is a collection 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. Unlike a mere pattern compilation or catalog, a pattern language includes rules and guidelines which explain how and when to apply its patterns to solve a problem which is larger than what any individual pattern can solve. These rules and guidelines suggest the order and granularity for applying each pattern in the language [2].

Intelligent Tutoring Systems incorporate built in expert systems in order to monitor the performance of a learner and to personalize instruction on the basis of adaptation to the learner's learning style, current knowledge level, and appropriate teaching strategies. Intelligent Tutoring Systems adapt themselves to the current knowledge stage of the learner and support different learning strategies on an individual basis [3]. The classical ITS architecture is composed of the following components:

- **Expert Module:** This module contains the domain knowledge.
- **Pedagogical (Tutor) Module:** This module provides the knowledge infrastructure necessary to tailor the presentation of the teaching material according to the student model.
- **Domain Module:** This model contains the knowledge about the actual teaching material.
- **Student Model:** This model stores details about the student's current problem-solving state and long-term knowledge progress, essential for adapting the material to the student's characteristics



- **Communication (User Interface) Module:** This module is responsible of interacting with the user.

The concept of patterns has received surprisingly little attention so far from researchers in the field of ITS. A recent analysis of a number of existing ITS architectures has revealed that many ITS designers and developers use their own solutions when faced with design problems that are common to different systems, models, and paradigms. However, a closer look into such solutions and their comparison often shows that different solutions and the contexts in which they apply also have much in common, just like the corresponding problems do. In all such cases we can talk of the existence of patterns [4].

In this research, we present **PLITS: A Pattern Language for Intelligent Tutoring Systems**. **PLITS** is not a mere collection of patterns that can be used in the implementation of Intelligent Tutoring Systems it includes rules and guidelines which explain how and when to apply its patterns to solve a problem which is larger than any individual pattern can solve. The patterns included in this pattern language were used in the implementation of the **Arabic Tutor** which is an Intelligent Language Tutoring System for teaching a subset of the basic rules of the Arabic Language. *The Arabic Tutor* was used and evaluated by students in the Arabic Language Institute in the American University in Cairo.

## 1.2 Research Motivation

Although numerous ITSSs have been developed to date, they are mostly used in research environments, and only a few have been used by large numbers of students in real classrooms.

**The main causes of such limited use of existing systems are the following:**

- Complexity of ITS development, and the difficulties with providing robust and flexible systems. The area is young; there are no well established methodologies or development tools. The time needed for the development of one hour of instruction in an ITS is estimated to be 100 hours of development time.
- ITSSs are often research products that are not accessible by many students.

- The hardware platforms available in most schools or universities are not the ones developers prefer, and porting systems between platforms is in no way a straightforward task [5].

**In this research we attempt to address all these obstacles by:**

- Creating a pattern language for intelligent tutoring systems by which we create a body of literature to help software developers resolve recurring problems encountered throughout all of software development process of any intelligent tutoring system. In this way, designers of new or existing ITS, especially inexperienced designers, can take advantage of previous design expertise and save precious time and resources.
- Using this pattern language in implementing a web based intelligent tutoring system that will benefit from both the accessibility of the web and the adaptivity of intelligent tutoring systems. Web-enabled versions of ITSs have the potential to reach a much wider audience as they face significantly fewer problems with hardware and software requirements.

**This research is important for two reasons:**

- It provides a unique pattern language that up to our knowledge wasn't approached by any other researcher which is a pattern language for intelligent tutoring system implementation.
- It implements an intelligent language tutoring system for teaching a subset of the Arabic language. We choose the domain of language learning because it is among the few domains that is gaining international popularity because of its importance to students internationally, irrespective of their mother tongue. Other ITS domains may have more restricted audience due to the dependence of the courses on the language they are written in. For example, a web-based course for the Arabic language may be used by students of any mother tongue who learn Arabic, whereas a web-based course for history written in German may only be used by history students who know German.

### 1.3 The Addressed Problem

Undoubtedly, the complexity of an ITS has become a critical issue that hampers the development process. The design and implementation of intelligent tutoring systems incorporates a variety of organizational, administrative, instructional and technological components [4]. The time needed for the development of one hour of instruction in an ITS is estimated to be 100 hours of development time [5].

This complexity can be overcome by creating a pattern language for intelligent tutoring systems that can help software developers resolve recurring problems encountered throughout all of software development process of any ITS. In this way, designers of new or existing ITS, especially inexperienced designers, can take advantage of previous design expertise and save precious time and resources.

The aim of this research is to increase the ITS community's awareness of the benefits of using patterns for development of ITSs by conducting an extensive analysis of existing Intelligent Tutoring Systems and their components in search for some possible common design decisions, common interactions among components, and common generalized principles underlying superficially different designs. More precisely, we are trying to extract patterns from numerous known examples, systems, architectures, designs, learning and teaching styles, strategies, etc.

We cannot talk of patterns in ITS only in the context of ITS architectures. On the contrary, there are many kinds of patterns in the way learners learn and in the way teachers teach, and all of them can be used as starting points when designing ITS. For example, patterns occur in speech acts of pedagogical interactions, in instructional design, in student modeling, and in teaching strategies.

We took this research one step further and created a pattern language for intelligent tutoring systems that include rules and guidelines which explain how and when to apply its patterns to

solve a problem which is larger than any individual pattern can solve. The patterns included in this pattern language were used in the implementation of the **Arabic Tutor** which is an Intelligent Language Tutoring System for teaching a subset of the basic rules of the Arabic Language. *The Arabic Tutor* was used and evaluated by students in the Arabic Language Institute in the American University in Cairo.

The aim of the **Arabic Tutor** is to benefit from the discovered patterns in its implementation and to produce an intelligent tutoring system that avoids all the problems facing traditional classrooms including [3]:

- 1) Multiple learners at the same time, e.g. in a classroom. Each of these learners will have a different depth of knowledge and thus will require different types of instruction.
- 2) These differences between learners change over time, since the change in depth of knowledge through instruction will vary from person to person.
- 3) In addition learners will have their preferred learning styles, and a typical lecture is limited to either repeating the same material over and over again in different formats trying to support different learning styles, or ignores these differences altogether.
- 4) In order to show learners whether they understood the material, tests are given to them. However, time limitations usually prevent the instructor from repeating the instruction to those individuals who failed parts of the test. Instead grades are used to inform the student that a particular knowledge type has not been mastered. At this point, the student alone is responsible for changing the situation and obtaining mastery, without the help of an instructor.

The **Arabic Tutor** will use the discovered patterns to reach a better teaching method by personalizing the instruction based on the background and the progress of each individual student.

This personalized method of instruction is characterized by [3]:

- 1) Learner controlled pacing.
- 2) The ability to retake tests until mastery is demonstrated.
- 3) Immediate feedback that is tailored according to the learner level and error made.
- 4) Small units of instructional material.
- 5) The use of proctors to administer feedback in the form of online testing and grading.
- 6) Providing learners with different versions, different difficulty levels, etc., based on their performance and previous knowledge.

#### **1.4 Outcomes and Contributions**

**The main contributions of this research include:**

1. **Formulating PLITS** - The first pattern language for intelligent tutoring systems implementation that includes rules and guidelines which explain how and when to apply its patterns to solve a problem which is larger than any individual pattern can solve.
2. **Implementing the Arabic Tutor** - an Intelligent Arabic Language Tutor over the World Wide Web for teaching a subset of the basic rules of the Arabic language was implemented that combines the flexibility and intelligence of Intelligent Tutoring Systems with the availability of the World Wide Web applications. This system is a proof of concept prototype; the system covers both the areas of intelligent tutoring systems and patterns for intelligent language tutoring systems.

#### **1.5 The Solution Approach**

This section highlights the main steps that were followed to reach the desired outcome.

##### **1.5.1 Phase One: Pattern Discovery**

We conducted an extensive survey on the existing Intelligent Tutoring Systems and their components in search for some possible common design decisions, common interactions among components, and common generalized principles underlying superficially different designs. More precisely, we tried to extract patterns from numerous known examples, systems, architectures, designs, learning and teaching styles, strategies, etc. This phase ended by discovering a number

of patterns that can be useful in intelligent tutoring system implementation.

The suggested patterns were used in more than one ITS and proved to be successful in their context. However, to prove the concept the suggested patterns had to be implemented and tested. Every pattern was implemented, tested, and documented. The patterns were developed on the suggested working environment. Class diagrams were developed based on the UML notations.

### 1.5.2 Phase Two: Formulating PLITS

**PLITS** is the first pattern language for intelligent tutoring systems implementation that includes rules and guidelines which explain how and when to apply its patterns to solve a problem which is larger than what any individual pattern can solve.

When we were formulating **PLITS** we wanted to provide a road map for any developer or designer to follow when faced by the problem of implementing an ITS. Thus we followed a comprehensive approach to cover all aspects related to ITS implementation. We investigated the use of design patterns, access patterns, instructional patterns, adaptive patterns, pedagogical patterns and interaction Patterns in ITS implementation. However when we were formulating **PLITS** we choose only the patterns that were relevant to ITS domain as illustrated in chapter 4 and 5.

### 1.5.3 Phase Three: Testing

This phase is divided into two steps:

#### 1.5.3.1 Testing PLITS (By Implementing the Arabic Tutor)

We aimed to build a proof of concept prototype by implementing the **Arabic Tutor** in order to prove that the suggested pattern language is useful and can act as a guideline for ITS implementation.

#### 1.5.3.2 Testing the Arabic Tutor

We went one step further and tested the **Arabic Tutor** in the Arabic language institute in the American University in Cairo. Statistics were gathered on the performance of students before

and after using our proposed intelligent tutoring systems in order to determine the efficiency of the system. A summary of the results of the **Arabic Tutor** Evaluation is found in section 5.6.

## 1.6 Thesis Organization

The rest of this document is organized as follows:

- **Chapter 2**

This chapter discusses the definition and fundamental concepts of reusability, patterns, anti-patterns, design patterns, frameworks, pattern languages, pattern catalogs and pattern systems. In addition, this chapter introduces the history, definition and components of intelligent tutoring systems, compares between intelligent tutoring systems and traditional tutoring systems, and discusses software training research approaches. This chapter also differentiates between standalone intelligent tutoring systems and intelligent tutoring systems on the World Wide Web. Finally, this chapter highlights the different architectures of intelligent tutoring systems on the World Wide Web and compares between them.

- **Chapter 3**

This chapter cites some of the related work that has been published by some researchers in the area of design patterns and intelligent language tutoring systems, it is divided into two parts; part one describes intelligent language tutoring systems related work and part two discusses design patterns in intelligent tutoring systems related work.

- **Chapter 4**

This chapter presents the proposed pattern language, and highlights how these patterns were used to implement the **Arabic Tutor**.

- **Chapter 5**

This chapter describes the development environment, features and architecture of the **Arabic Tutor**. It also presents the test environment and the test results.

- **Chapter 6**

This chapter concludes the research and gives some future directions that are related to the research.

## Chapter 2

# Research Background

### 2.1 Reusability

Reusability is best described as the degree to which a software module or other work product can be used in more than one computing program or software system. Using previously developed and tested components when building an application saves development efforts and reduces the risk of introducing errors in the system [6].

### 2.2 Design Patterns

Design patterns can be defined as descriptions of communicating objects and classes that are customized to solve a general design problem in a particular context [7].

A design pattern names, abstracts, and identifies the key aspects of a common design structure that make it useful for creating a reusable object-oriented design. The design pattern identifies the participating classes and their instances, their roles and collaborations, and the distribution of responsibilities. Each design pattern focuses on a particular object-oriented design problem or issue. It describes when it applies, whether or not it can be applied in view of other design constraints, and the consequences and trade-offs of its use. Since we must eventually implement our designs, a design pattern also provides sample code to illustrate an implementation. Although design patterns describe object-oriented designs, they are based on practical solutions that have been implemented in mainstream object-oriented programming languages [7].

#### 2.2.1 What is a pattern?

In [8], a pattern was defined as an abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts.

The above definition points out that, within the software patterns community, the notion of a pattern is "geared toward solving problems in design." More specifically, the concrete form which



recurs is that of a solution to a recurring problem. But a pattern is more than just a battle-proven solution to a recurring problem. The problem occurs within a certain context, and in the presence of numerous competing concerns. The proposed solution involves some kind of structure which balances these concerns, or "forces", in the manner most appropriate for the given context. Using the pattern form, the description of the solution tries to capture the essential insight which it embodies, so that others may learn from it, and make use of it in similar situations. The pattern is also given a name, which serves as a conceptual handle, to facilitate discussing the pattern and the jewel of information it represents. So a definition which more closely reflects its use within the patterns community is:

Alexander describes it a bit more colorfully as:

Each pattern is a three-part rule, which expresses a relation between a certain context, a problem, and a solution [9].

### 2.2.2 Features of Good Patterns

- It solves a problem:** Patterns capture solutions, not just abstract principles or strategies.
- It is a proven concept:** Patterns capture solutions with a track record, not theories or speculation.
- The solution isn't obvious:** Many problem-solving techniques (such as software design paradigms or methods) try to derive solutions from first principles. The best patterns generate a solution to a problem indirectly -- a necessary approach for the most difficult problems of design.
- It describes a relationship:** Patterns don't just describe modules, but describe deeper system structures and mechanisms.
- The pattern has a significant human component;** all software serves human comfort or quality of life; the best patterns explicitly appeal to aesthetics and utility [10].

### 2.2.3 Anti-Patterns

If a pattern represents a "best practice", then an **anti-pattern** represents a "lesson learned." There are two notions of "anti-patterns".

1. Those that describe a bad solution to a problem which resulted in a bad situation.
2. Those that describe how to get out of a bad situation and how to proceed from there to a good solution [11].

### 2.2.4 Pattern Classification

#### Architectural Patterns

An *architectural pattern* expresses a fundamental structural organization or schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationships between them [12].

#### Design Patterns

A *design pattern* provides a scheme for refining the subsystems or components of a software system, or the relationships between them. It describes a commonly recurring structure of communicating components that solves a general design problem within a particular context [12].

#### Idioms

An *idiom* is a low-level pattern specific to a programming language. An idiom describes how to implement particular aspects of components or the relationships between them using the features of the given language [12].

The difference between these three kinds of patterns is in their corresponding levels of abstraction and detail. Architectural patterns are high-level strategies that concern large-scale components and the global properties and mechanisms of a system. They have wide-sweeping implications which affect the overall skeletal structure and organization of a software system. Design patterns are medium-scale tactics that flesh out some of the structure and behavior of entities and their relationships. They do *not* influence overall system structure, but instead define *micro-architectures* of subsystems and components. Idioms are paradigm-specific and

language-specific programming techniques that fill in low-level internal or external details of a component's structure or behavior [12].

Another way of classifying patterns that partitions the different kinds of patterns among analysis, design and implementation can be pointed out as follows:

### **Conceptual Patterns**

A *conceptual pattern* is a pattern whose form is described by means of terms and concepts from an application domain [8].

### **Design Patterns**

A *design pattern* is a pattern whose form is described by means of software design constructs, for example objects, classes, inheritance, aggregation and use-relationship [8].

### **Programming Patterns**

A *programming pattern* is a pattern whose form is described by means of programming language constructs [8].

Using these definitions, conceptual patterns are based upon metaphors in a restricted application domain. Design patterns complement, or elaborate upon conceptual patterns by delving into the implementation of elements from the conceptual space. And programming patterns descend further into implementation details using a specific implementation language [8].

When comparing and contrasting these two sets of definitions, it appears that programming patterns are equivalent to idioms. For the other types of patterns described above, the first set of authors choose to delineate them by their architectural scope whereas the latter set of authors choose to delineate them by whether they employ language from the problem space or the solution space[8].

#### **2.2.5 Elements of a Pattern**

Despite the use of these differing pattern formats, it is generally agreed that a pattern should contain certain essential elements. Regardless of the particular format/headings used (or lack thereof) a pattern should normally be illustrated using a simple diagram and should be written

using a narrative writing style.

Each pattern should have a name; it should also have the following information [7]:

- **Intent**

A short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design or issue does it address?

- **Applicability**

What are the situations in which the design pattern can be applied? What are examples of poor designs that the pattern can address? How can you recognize these situations?

- **Structure**

The structure is a graphical representation of the classes in the pattern.

- **Participants**

The participants are the classes and/or objects participating in the design pattern and their responsibilities.

- **Consequences**

How does the pattern support its objectives? What are the tradeoffs and results of using the pattern? What aspect of system structure does it let you vary independently?

- **Implementation**

What pitfalls, hints or techniques should you be aware of when implementing the pattern? Are there language specific issues?

- **Sample code**

Code fragments that illustrate how you might implement the pattern.

- **Related Patterns**

What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?

- **Anti-patterns (optional)**

Solutions that are inferior or do not work in this context. The reason for their rejection should be explained. The anti-patterns may be valid solutions in other contexts, or may never be valid.

## • References

The references represent acknowledgements of those who developed or inspired the pattern.

### 2.2.6 Design Patterns and Frameworks

A **framework** is a set of cooperating classes that make up a reusable design for a specific class of software. A framework provides architectural guidance by partitioning the design into abstract classes and defining their responsibilities and collaborations. A developer customizes a framework to a particular application by sub classing and composing instances of framework classes [8].

A framework dictates the architecture of your application. It will define the overall structure, its partitioning into classes and objects, the key responsibilities thereof, how the classes and objects collaborate, and the thread of control. A framework predefines these design parameters so that you, the application designer/implementer, can concentrate on the specifics of your application. The framework captures the design decisions that are common to its application domain. Frameworks thus emphasize *design reuse* over code reuse; though a framework will usually include concrete subclasses you can put to work immediately [8].

The difference between a framework and an ordinary programming library is that a framework employs an *inverted flow of control* between itself and its clients. When using a framework, one usually just implements a few callback functions or specializes a few classes, and then invokes a single method or procedure. At this point, the framework does the rest of the work for you, invoking any necessary client callbacks or methods at the appropriate time and place [8].

The major differences between design patterns and frameworks can be described as follows [8]:

1. *Design patterns are more abstract than frameworks.* Frameworks can be embodied in code, but only *examples* of patterns can be embodied in code. One of the strengths of frameworks is that they can be written down in programming languages and not only studied but executed and reused directly. In contrast, design patterns have to be implemented each time they are used. Design patterns also explain the intent, trade-offs,

and consequences of a design.

2. *Design patterns are smaller architectural elements than frameworks.* A typical framework contains several design patterns but the reverse is never true.
3. *Design patterns are less specialized than frameworks.* Frameworks always have a particular application domain. In contrast, design patterns can be used in nearly any kind of application. While more specialized design patterns are certainly possible, even these wouldn't dictate application architecture.

### 2.2.7 Pattern Languages

A pattern Language is a collection of patterns that form a vocabulary for understanding and communicating ideas. Such a collection may be skillfully woven together into a cohesive "whole" that reveals the inherent structures and relationships of its constituent parts toward fulfilling a shared objective. If a pattern is a recurring solution to a problem in a context given by some forces, then 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[2].

In [10] a pattern language is defined as follows:

A pattern language defines a collection of patterns and the rules to combine them into an architectural style. Pattern languages describe software frameworks or families of related systems.

In [9] remarks were made on what a pattern language truly embodies, it states the following:

- A pattern language is generative. It not only tells us the rules of arrangement, but shows us how to construct arrangements as many as we want which satisfy the rules.
- A pattern language gives each person who uses it, the power to create an infinite variety of new and unique buildings, just as his ordinary language gives him the power to create

an infinite variety of sentences.

- The structure of the language is created by the network of connections among individual patterns.
- Each pattern depends both on the smaller patterns it contains and on the larger patterns within which it is contained.
- The language is a good one, capable of making something whole, when it is morphologically and functionally complete.

In [2] it is mentioned that pattern languages provide a dynamic process for the orderly resolution of problems within their domain which indirectly leads to the resolution of a much broader problem. The patterns and rules in a pattern language combine to form an architectural style. In this manner, pattern languages guide system analysts, architects, designers, and implementers to produce workable systems that solve common organizational and development problems at all levels of scale and diversity.

#### 2.2.7.1 Pattern Language General Model

A formal description of patterns makes it less ambiguous for the parties involved to decide what a pattern is supposed to look like, in terms of structure and content. It also makes it possible to design computer-based tools that help authors in writing, and readers in understanding patterns [13].

##### Formal syntactic definition:

- A *pattern language* is a directed acyclic graph (DAG)  $PL = (\mathbf{P}, \mathbf{R})$  with nodes  $\mathbf{P} = \{P_1, P_n\}$  and edges  $\mathbf{R} = \{R_1, R_m\}$ .
- Each node  $P \in \mathbf{P}$  is called a *pattern*.
- For  $P, Q \in \mathbf{P}$ :  $P$  references  $Q \iff \exists R = (P, Q) \in \mathbf{R}$ .
- The set of edges leaving a node  $P \in \mathbf{P}$  is called its *references*. The set of edges entering it is called its *context*.

- Each node  $P \in \mathbf{P}$  is itself a set  $P = \{n, r, i, p, f_1 \dots f_i, e_1 \dots e_j, s, d, \}$  of a name  $n$ , ranking  $r$ , illustration  $i$ , problem  $p$  with forces  $f_1 \dots f_i$ , examples  $e_1 \dots e_j$ , the solution  $s$ , and diagram  $d$  [13].

### 2.2.7.2 A Pattern Language Example

Different types of relationships exist between patterns. Welie [14] identified three main relationships:

- **Aggregation**

Patterns can aggregate or include other patterns. This is a form of "has a" relationship.

- **Specialization**

This relationship is resembled when a pattern is a more specific version of another pattern. This is a form of "is a" relationship.

- **Association**

This relationship is resembled when a pattern is associated to other patterns because they occur in the larger context of the problem, or because the patterns are alternatives for the same kind of problems. This is not a "has a" or "is a" relationship but simple "related to" relationship. These all different relationships can all exist within one hierarchical structure forming the pattern language [14].

When organizing patterns into a hierarchy to form a pattern language, the scale hierarchy of the problems addressed by the patterns is considered. This implies that patterns will be available at different levels of a top down design approach. Patterns thus require to be organized into levels as well [14].

Welie [14] proposed an organization of patterns into the following four levels:

- **Posture level patterns:** Every site or application has a purpose for its existence, for commercial sites there are usually business goals to be achieved while other sites have more personal or social goals. These patterns are written to describe the requirements for creating applications with a certain purpose.



- **Experience level patterns:** Experiences are the high level goals for which the user uses the application. These patterns describe common experiences and specify which lower level patterns can be used to create these experiences.
- **Task level patterns:** These patterns describe a series of interactions on one or more objects for solving a problem. They are mainly domain independent patterns. The posture and experience patterns set the context specifics and the task patterns are used to fill in the blanks. Task patterns can often be "drawn" using flow diagrams and sketches.
- **Action level patterns:** These are the lowest level patterns. They are not related to any specific user goal. They describe solutions which are usually uses of known widgets or descriptions of custom made widgets.

Figure (2-1) shows a partial pattern language, which uses the previously mentioned pattern language organization.

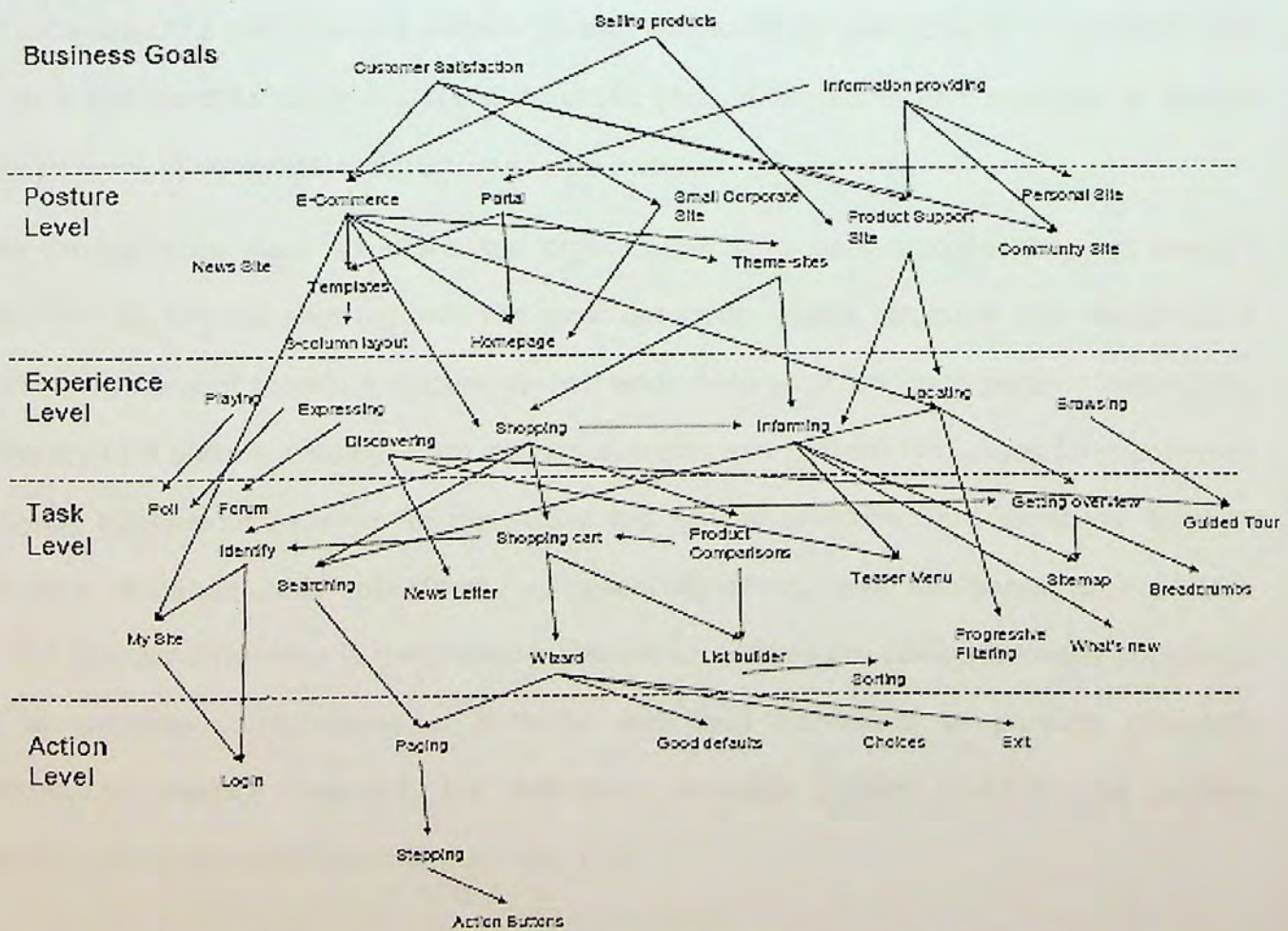


Figure (2-1): A Partial Pattern Language for Web Design (Centered Around Shopping) [14]

### 2.2.8 Pattern Catalogs and Systems

In [12] different kinds of pattern collections that possess varying degrees of structure and interaction have been classified into pattern catalogs, systems, and languages:

#### Pattern Catalogs

A *pattern catalog* is a collection of related patterns (perhaps only loosely or informally related). It typically subdivides the patterns into at least a small number of broad categories and may include some amount of cross referencing between patterns.

#### Pattern Systems

A *pattern system* is a cohesive set of related patterns which work together to support the construction and evolution of whole architectures. Not only is it organized into related groups and subgroups at multiple levels of granularity, it describes the many interrelationships between the patterns and their groupings and how they may be combined and composed to solve more complex problems. The patterns in a pattern system should all be described in a consistent and uniform style and need to cover a sufficiently broad base of problems and solutions to enable significant portions of complete architectures to be built.

A pattern catalog adds more structure and organization to a pattern collection, but doesn't usually go very far beyond showing only the most outwardly visible structure and relationships (if in fact it shows any of them). A pattern system adds deep structure, rich pattern interaction, and uniformity to a pattern catalog. Both pattern systems and pattern languages form coherent sets of tightly interwoven patterns for describing and solving problems in a particular domain. But a pattern language adds robustness, comprehensiveness, and wholeness to a pattern system. The primary difference is that, ideally, pattern languages are computationally complete, showing all possible combinations of patterns and their variations to produce complete architectures. In practice however, the difference between pattern systems and pattern languages can be extremely difficult to ascertain [12].

While a pattern system may be a cohesive collection of patterns about a very broad topic, a pattern language has to be about more than just a "broad topic". A pattern language ultimately

corresponds to a single-minded collection that forms a kind of "mega pattern" or "super pattern" in that the entire language possesses an underlying, shared problem and its context, forces, solution, resulting context, and rationale. This coherence of purpose is what gives the pattern language a sense of closure. A pattern system does not necessarily have *all* of these pattern elements. It may focus on an equally broad or narrow topic, but it may not necessarily have a clear mission or agenda (and may result in many relationships between patterns being harder to find), or it leaves several important gaps unfilled in the problem space (and hence may not attain overall resolution or closure) [12].

The most important difference between pattern languages and pattern systems is that pattern languages are not created all at once, they evolve from pattern systems through the process of piecemeal growth and a pattern system may, in turn, evolve from a pattern catalog in a similar manner) [12].

### 2.9 What is an Intelligent Tutoring System?

Intelligent Tutoring Systems incorporate built in expert systems in order to monitor the performance of a learner and to personalize instruction on the basis of adaptation to a learner's learning style, current knowledge level, and appropriate teaching strategies [3].

Intelligent Tutoring Systems adapt themselves to the current knowledge stage of the learner and support different learning strategies on an individual basis [3].

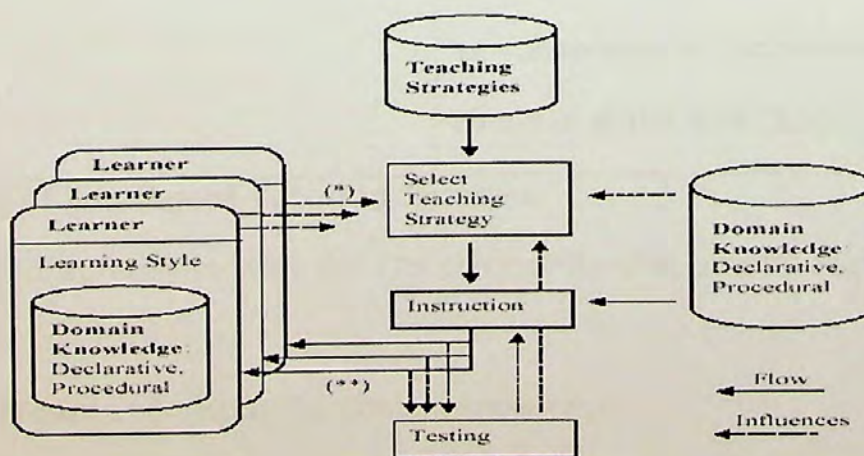


Figure (2-2): Need for Personalized Instruction [3]

Figure (2-2) shows an overview of the instruction process. A teaching Strategy is selected based on the knowledge type being taught and the learner's current depth of knowledge. The actual instruction then changes the learner's level of knowledge depth by either adding new knowledge or deepening existing knowledge. Testing is conducted to evaluate the success of the instruction and the potential need for a repetition or modification of instruction [3].

### 2.10 Intelligent Tutoring Systems versus Traditional Tutoring Systems

**Table (1): ITS and Traditional TS Comparison**

<b>Intelligent Language Tutoring Systems</b>	<b>Traditional Tutoring Systems</b>
Multimedia capabilities can be used to achieve a more varied, authentic and contextualized learning environment.	Traditional workbook exercises are used, making little use of the multimedia capabilities.
Provides flexibility in handling student answers.	From a pedagogical perspective, the definition of acceptable answers to exercises is highly constrained.
Provides error-specific feedback	Error feedback commonly does not address the source of an error. The system's feedback is a generic, catch-all response. In addition to the pedagogical limitations, the student has to consult the textbook which is an unnecessary inconvenience given the potential of the Web [15].

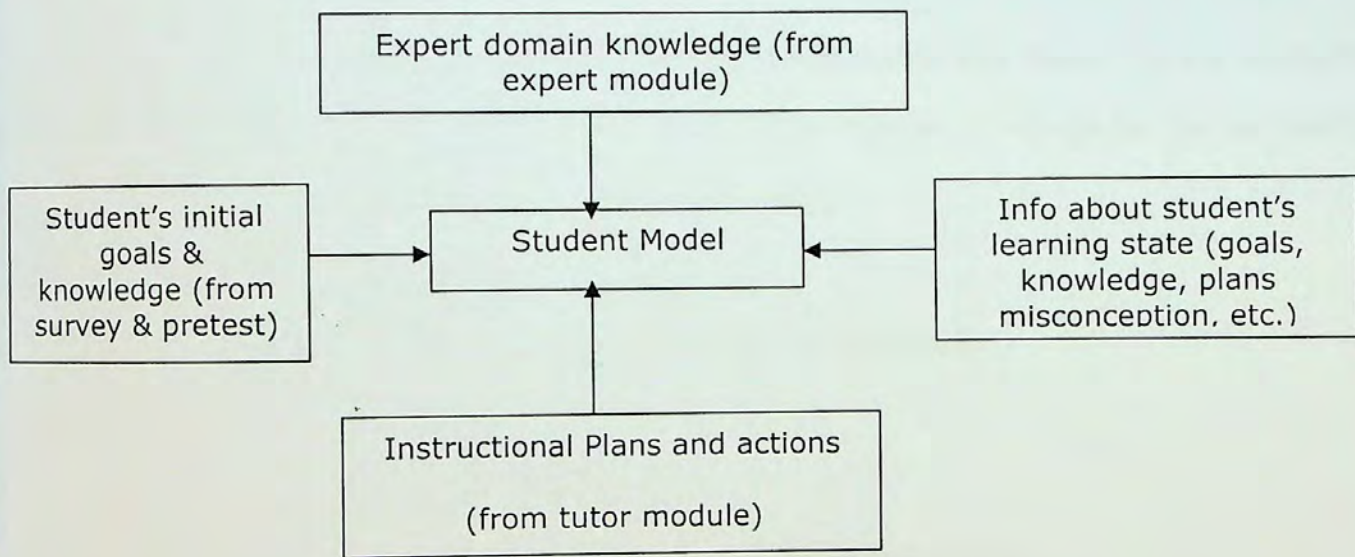
### 2.11 Components of Intelligent Tutoring Systems

There's a widespread agreement with the ITS community that an ITS architecture consists of four modules [16]:

1. **Expert Module:** Contains the domain knowledge.
2. **Student Module:** Diagnosis what the student knows.

3. **Tutor Module:** Identifies which deficiencies in knowledge to focus on and selects the appropriate instructional strategies to present that knowledge.
4. **User Interface:** That is responsible for communication with the learner.

The learner interacts with both the tutoring module to receive instructions, feedback and guidance and the expert module to solve problems and look at examples. Student modeling is the key to individualized knowledge base instruction. The student model allows the tutoring system to adapt to the needs of individual learners. The student model uses information provided from the user, expert module, an instructional module, to keep an accurate model of the learners' knowledge level and capabilities and also to guide the constructional Strategy [16].



**Figure (2-3): The Student Model [16]**

### 2.12 History of Intelligent Tutoring Systems

Computer Based Training (CBT) has become more popular since the 1990s. Corporations delivered 10% of their training through CBT systems in 1996, and 16% of the fortune 1000 companies trained through multimedia systems [3].

Traditional classrooms and conventional CBT systems faced a number of problems [3]:

- 1) Multiple learners at the same time, e.g. in a classroom. Each of these learners will have a different depth of knowledge and thus will require different types of instruction.
- 2) Differences between learners change over time, since the change in depth of

knowledge through instruction will vary from person to person.

- 3) Learners have their preferred learning styles, and a typical lecture is limited to either repeating the same material over and over again in different formats trying to support different learning styles, or ignores these differences altogether.
- 4) Tests are given to show learners whether they understood the material. However, time limitations usually prevent the instructor from repeating the instruction to those individuals who failed parts of the test. Instead grades are used to inform the student that a particular knowledge type has not been mastered. At this point, the student alone is responsible for changing the situation and obtaining mastery, without the help of an instructor.

A better instruction method would be to personalize the instruction based on the background and the progress of each individual student via either human or computer based tutors. A personalized method of instruction is characterized by [3]:

- 1) Learner controlled pacing.
- 2) The ability to retake tests until mastery is demonstrated.
- 3) Immediate feedback.
- 4) Small units of instructional material.
- 5) The use of peer proctors to administer feedback and testing.
- 6) Optional lectures.

For computer based training systems, the computer can take over the role of peer proctors in form of online testing and grading, and lectures can be placed online as well. The other principles can be directly applied to the development of ITS by providing learners with different versions, different difficulty levels, etc., based on their performance and previous knowledge [3].

## 2.13 Architecture of Intelligent Tutoring Systems

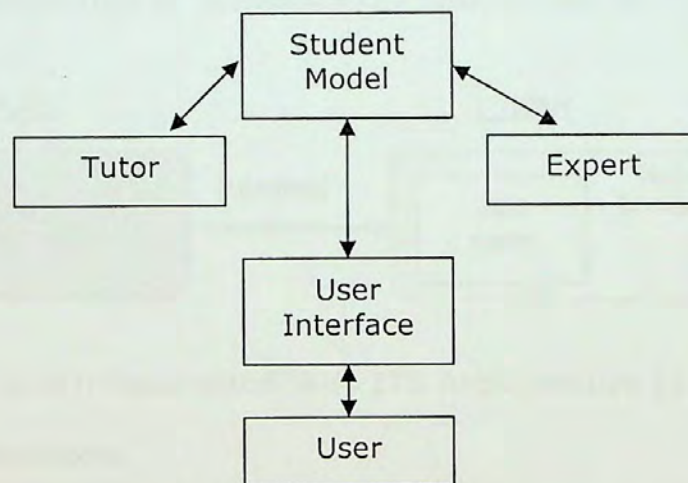
### 2.13.1 Architecture of Standalone Intelligent Tutoring Systems

There's a widespread agreement with the ITS community that an ITS architecture consists of four modules [16]:

1. **Expert Module:** Contains the domain knowledge.

2. **Student Module:** Diagnosis what the student knows.
3. **Tutor Module:** Identifies which deficiencies in knowledge to focus on and selects the appropriate instructional strategies to present that knowledge.
4. **User Interface:** That is responsible for communication with the learner.

The learner interacts with both the tutoring module to receive instructions, feedback and guidance and the expert module to solve problems and look at examples. Student modeling is the key to individualized knowledge base instruction. The student model allows the tutoring system to adapt to the needs of individual learners. The student model uses information provided from the user, expert module, an instructional module, to keep an accurate model of the learners' knowledge level and capabilities and also to guide the constructional Strategy [16].



**Figure (2-4): Standalone ITS Architecture [16]**

### 2.13.2 Architectures of Intelligent Tutoring Systems on the World Wide Web

Several architectures for Web-enabled tutors have emerged so far, all based on the client-server architecture. These architectures may be classified into the following three categories[5]:

#### 2.13.2.1 Centralized Architecture

This architecture consists of the following components:

- **Web Server** : Run on the server side.
- **Application Server**: Run on the server side and performs all tutoring functions.
- **Student Interface**: Displayed in a Web browser on the client's machine. It consists of a set of HTML pages.

The student interacts with HTML entry forms, and the information is sent to the Web server, which passes the student's requests and actions to the application server[5].

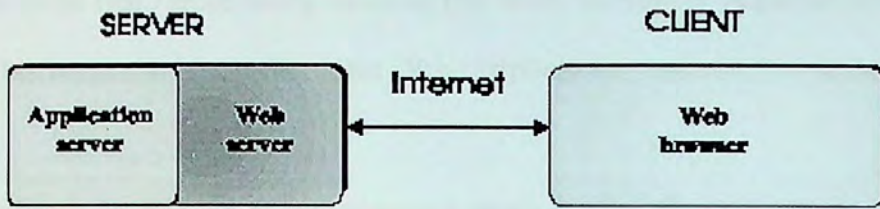


Figure (2-5): Centralized Web ITS Architecture [5]

2.13.2.2 Replicated Architecture

The entire tutor resides in a Java applet that needs to be downloaded and is executed on the student's machine. All tutoring functions are therefore performed on the client's machine, while the server is only used as a repository of software to be downloaded[5].

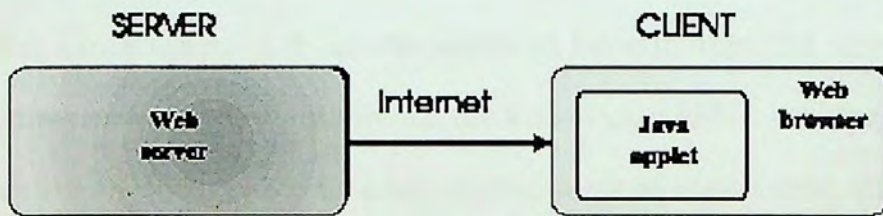


Figure (2-6): Replicated Web ITS Architecture [5]

2.13.2.3 Distributed Architecture

The tutoring functionality is distributed between the client and the server. The exact policy on distributing the functions may vary. Most often, the application server consists of a student modeler, which creates and maintains student models for all users, a domain module, capable of solving and/or selecting problems, and a pedagogical module. The user interface is usually Java-based and may perform some teaching functions[5].

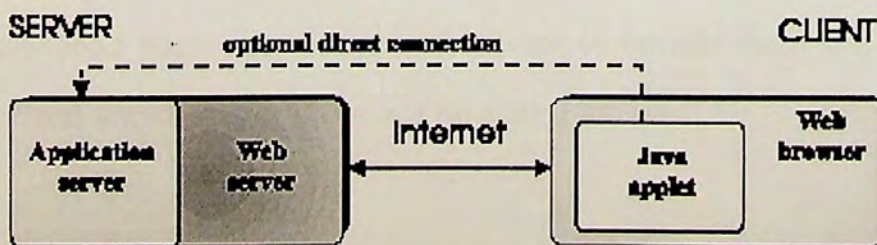


Figure (2-7): Distributed Web ITS Architecture [5]



Additional functionality in the interface includes immediate feedback for each problem solving step, and interactive graphics and simulations. Communication between the interface and the application server does not necessarily involve the Web server; it is possible to establish a direct TCP connection between the applet and the application server in order to speed up the system[5].

#### **2.13.2.4 Comparison Between the Different Web Based Architectures**

The amount of effort involved in building a tutor with a replicated architecture is the same as building a standalone system. These systems are very fast, as all processing is done on the client's machine[5].

However, a significant limitation of this architecture is the fact that the student model is stored on the machine where the tutor has been executed. Therefore, the student always needs to use the system from the same machine if he/she wants to benefit from the summaries of previous sessions stored in the student model, otherwise the knowledge about previous sessions would be lost and the system would not be able to adapt to the student easily. One interesting solution to this problem may be found in ADELE, where copies of student models are also kept on the server between sessions for persistent storage. Although this solution removes the requirement that a student always has to use the tutor from the same machine, there is still a problem if a network error occurs before the student completes a session, as the most recent information about student's performance will then be lost[5].

In replicated and distributed architectures it is necessary for a student to download software in order to start using a system - a task that some students may find frustrating. Furthermore, it is necessary to download each new release of a tutor to benefit from the improvements. In the case of a centralized architecture, there are no such problems[5].

A significant advantage of the centralized and distributed architectures is the fact that all student models are kept in one place (on the server) and the student can use the system from any

machine[5].

Additional knowledge structures, needed by the expert or pedagogical module, may be shared. A problem with these two architectures may be the reduced speed and the high network traffic, caused by communications between the client and the server. The situation might be better for a system with distributed architecture, as some of the tutoring actions are performed on the client side and hence the number of communications is reduced. However, communicating between the interface and the server in a distributed architecture may require special techniques, which introduces additional complexity to system development[5].

### **2.13.3 Web based Intelligent Tutoring Systems versus Standalone Intelligent Tutoring Systems**

Web based ITS offer several advantages in comparison to standalone ITS[5]:

1. They minimize the problems of distributing software to users and hardware/software compatibility.
2. New releases of tutors are immediately available to everyone.
3. More importantly, students are not constrained to use specific machines in their schools, and can access Web-enabled tutors from any location and at any time.

## Chapter 3

### Related Work

Indeed, the domain of language learning is probably among the few domains that may be of interest to students internationally, irrespective of their mother tongue.

Other ITS domains may have more restricted audience due to the dependence of the courses on the language they are written in. For example, a web-based course for the French language may be used by students of any mother tongue who learn French, whereas a web-based course for history written in Greek may only be used by history students who know Greek. This is an additional reason that advocates in favor of further research and development in the area of web-based ICALL (Intelligent Computer Aided Language Learning) [17].

In our search for patterns that could be used in ITS implementation we aimed to implement a prototype ITS that can act as a proof of concept. We chose to implement the **Arabic Tutor** which is an ITS for teaching a subset of the Arabic Language.

In section 3.1 we made a comparison between Standalone (non web based) Intelligent Language Tutoring Systems (ILTS). We concentrated on the ILTS because our prototype (**The Arabic Tutor**) also deals with the domain of language learning.

### 3.1 Standalone Intelligent Language Tutoring Systems

**Table (2): Standalone Intelligent Language Tutoring Systems**

	<b>The German Tutor[15]</b>	<b>An Interactive Course Support System for Greek[18]</b>	<b>CAPIT: An ITS for Capitalization and Punctuation[19]</b>	<b>The Verb Expert[20]</b>
<b>System Goal</b>	German vocabulary and grammar practice	Greek vocabulary and grammar practice	English capitalization and punctuation practice	English verbs practice
<b>Audience</b>	Learners of German	English Learners of Greek	Learners of English	Italian learners of English
<b>Learner Levels</b>	Beginner Intermediate Advanced	Beginner Intermediate Advanced	N/A	N/A
<b>Features</b>	Error-specific feedback according to the learner level  Individualization of the learning process	Error-specific feedback according to the learner level  Individualization of the learning process	Error-specific feedback according to violated constraint  Individualization of the learning process  Animated images and Current score is used to motivate students	Error-specific feedback according to violated constraint  Individualization of the learning process
<b>System Architecture</b>	<b>Natural Language Processing component (NLP)</b> to analyze ill-formed sentences, which consists of a grammar checker and a parser  <b>Student Model</b>  <b>Spelling checker</b> and a <b>syntactic parser</b> which detect spelling, syntactical, and morphological errors	<b>NLP component</b>  <b>Student Model</b>	<b>Database of constraints</b> specifying the correct patterns of punctuation and capitalization  <b>Database of problems</b>  <b>Student models</b> which determines which constraints are relevant to the current solution, and which constraints are satisfied  <b>User interface</b>  <b>Pedagogical module</b> which is responsible for choosing next problem and specific feedback messages	<b>Domain expert</b> which generates the right tense for the verbs appearing in the exercises presented to the student  <b>Tutor Module</b> which manages the teaching activity and the interaction with the student and chooses the next topic to be taught according to student competence level.  <b>The Student Model</b> which evaluates the student's knowledge level and forms a hypothesis on the reason for the user error
<b>Exercises</b>	Dictation, Build a Phrase, Which Word is Different, Word Order Practice, Fill-in-the-Blank, Build a Sentence	Guess the Word, Find the Word, Which Word is Different, Word Order Practice	Completion, Check-and correct	Fill in the blanks with the right tense of verb.

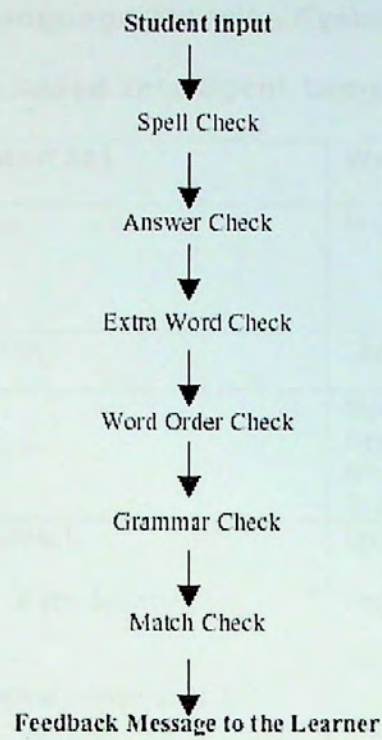


Figure (3-1): NLP Modules [18]

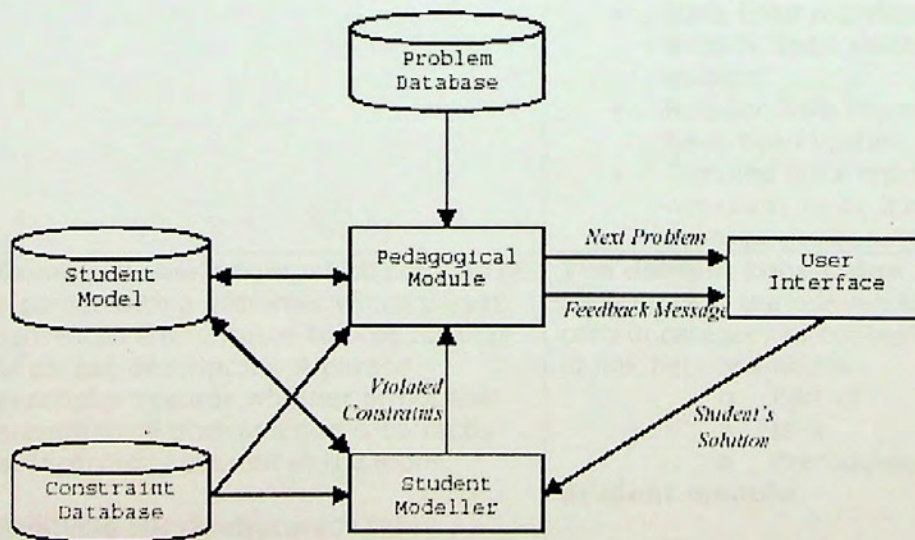


Figure (3-2): CAPIT Architecture [19]

In section 3.2 we show a comparison between Web Based Intelligent Language Tutoring Systems. We concentrated on the ILTS because our prototype (**The Arabic Tutor**) also deals with the domain of language learning.

### 3.2 Web Based Intelligent Language Tutoring Systems

**Table (3): Web Based Intelligent Language Tutoring Systems**

	<b>The German Tutor[21]</b>	<b>Web Passive Voice Tutor(Web PVT)[17]</b>
<b>System</b>	Grammar practice	English passive voice practice.
<b>Goal</b>		
<b>Audience</b>	Learners of German	Learners of English
<b>Learner Levels</b>	Beginner Intermediate Expert	Novice Beginner Intermediate Expert
<b>Features</b>	<p>Error specific feedback</p> <p>Individualization of the learning process.</p> <p>Records grammatical constraints that are met or not.</p> <p>Analysis of students' language input.</p>	<p>Error specific feedback</p> <p>Individualization of the learning process.</p> <p>Intelligent analysis of students' solutions</p> <p>Mal-rules related to language transfer are used in case the mother tongue of a student is known.</p> <p>Adaptive navigation support</p> <ul style="list-style-type: none"> <li>• Bold links, represent concepts that are ready to be learned and are recommended by the system.</li> <li>• Italic links represent concepts that have already been visited and are known to the student.</li> <li>• Regular links represent concepts that have been visited.</li> <li>• Dimmed links represent concepts that are not ready to be learned due to prerequisite occurrence.</li> </ul>
<b>System Architecture</b>	<p><b>Domain Knowledge:</b> which consists of a parser with a grammar which parses sentences and phrases to produce sets of phrase descriptors. A phrase descriptor records whether or not the grammatical phenomenon is correctly or incorrectly present in the input.</p> <p><b>Analysis Hierarchy:</b> which takes a phrase descriptor as input and generates sets of possible responses to the learner's input.</p> <p><b>Student Model</b></p> <p><b>Filtering Module:</b> Which selects the desired parse and decides on the order of the feedback messages.</p>	<p><b>The domain knowledge</b></p> <p>Each node in the domain knowledge represents a certain category of concept. There are three kinds of link between nodes:</p> <ul style="list-style-type: none"> <li>o Part-of</li> <li>o Is-a</li> <li>o Prerequisite</li> </ul> <p><b>Student module</b></p> <p><b>Tutor module:</b> That consults the student model in order to provide adaptive navigation support and to individualize the feedback.</p> <p><b>User interface module:</b> That consists of a set of dynamically constructed HTML pages and forms that is chosen by the tutor module.</p>
<b>Exercises</b>	<ul style="list-style-type: none"> <li>• Dictation</li> <li>• Build a Phrase</li> <li>• Which Word is Different</li> <li>• Word Order Practice</li> <li>• Fill-in-the-Blank</li> <li>• Build a Sentence</li> </ul>	Rewrite in another voice(active or passive)

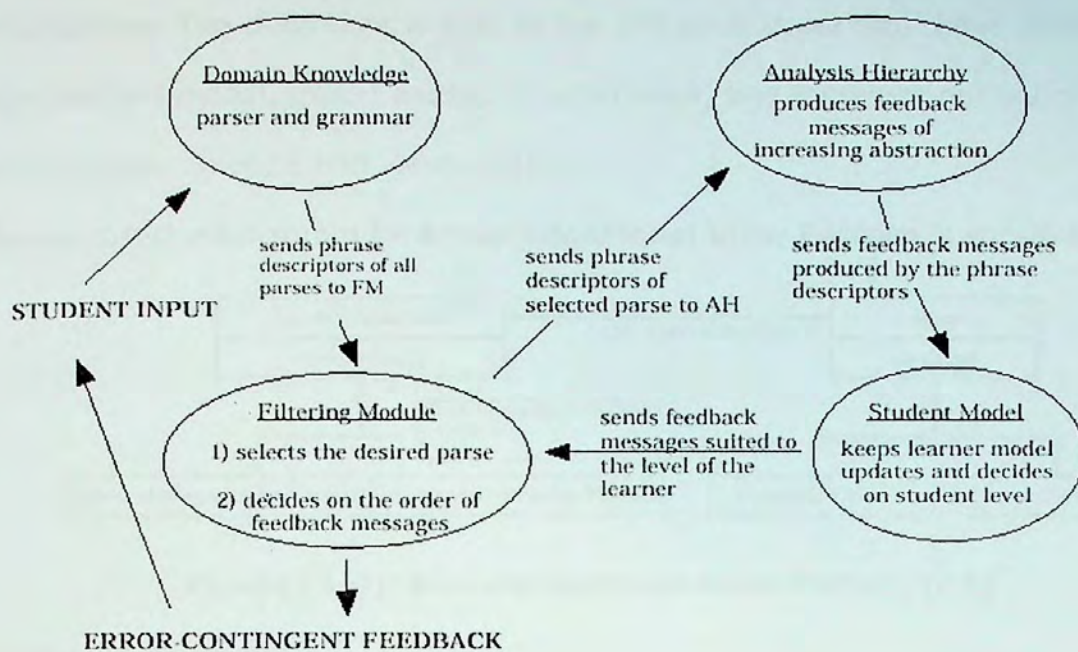


Figure (3-3) : The German Tutor[21]

### 3.3 Design and Analysis Patterns in ITS

#### 3.3.1 KnowledgeModel View Pattern

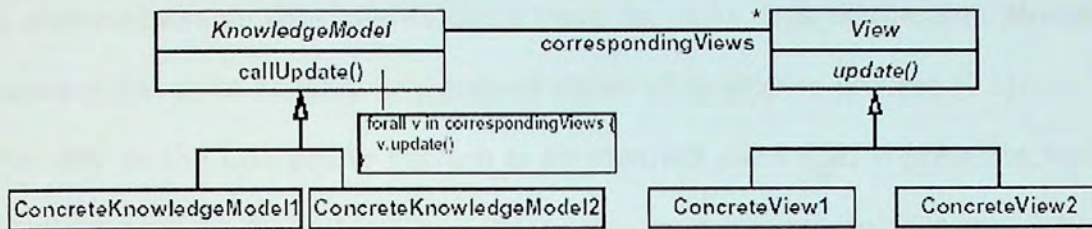
In [22] an architectural pattern for ITS that generalizes two architecture proposals was introduced, the "integration-oriented" architecture from Peter Brusilovsky and an alternative proposal from Meike Gonschorek. This general pattern was called "Knowledge Model View" pattern as a reference to the general architectural pattern "Model-View-Controller" and "Document-View", because there are similarities between this and the general architectural patterns.

KnowledgeModel View means that the architecture has distinct components for:

- a) All kinds of *knowledge models* within an ITS.
- b) Components for presentation of information for the user and interaction between user and system (these can be called *views*). Views are updated when the content of a knowledge model changes, so a typical interaction protocol exists between models and views, which is called "Publish-Subscribe". So the basic structure of this ITS-pattern is very similar to the general software engineering pattern "Document-View", consisting of one central document that contains all the relevant knowledge/data, and several views, which are updated every time the

document changes. The difference is that in the ITS-context we may have multiple sources of knowledge (student model, expert model, tutoring rules) and therefore not a  $1:n$  relation but an  $m:n$  relation between models and views [22].

The typical structure diagram for **KnowledgeModel View Pattern** is shown in figure (3-4):



**Figure (3-4): KnowledgeModel View Pattern [22]**

### 3.3.2 Composite Pattern

**Classification and Intent.** Composite is a structural pattern. Its intent is to compose objects into tree structures to represent part-whole hierarchies. It lets clients treat individual objects and compositions of objects uniformly [23].

**Motivation.** Lesson presentation planner of an ITS may decide to build an agenda of the topics to be presented during the lesson. Complex topics can be divided into simple elements, like concepts, text, graphics, and the like, or into a sequence of subtopics (simpler topics). Each subtopic in turn can be further subdivided into a lower level sequence of simple elements and other subtopics, producing an agenda like in the following example [23].

#### 1. Topic 1

Text

Graphics

Concept A

##### 1.1. Subtopic 1.1

##### 1.2. Subtopic 1.2

Text

Concept B

##### 1.2.1. Subtopic 1.2.1

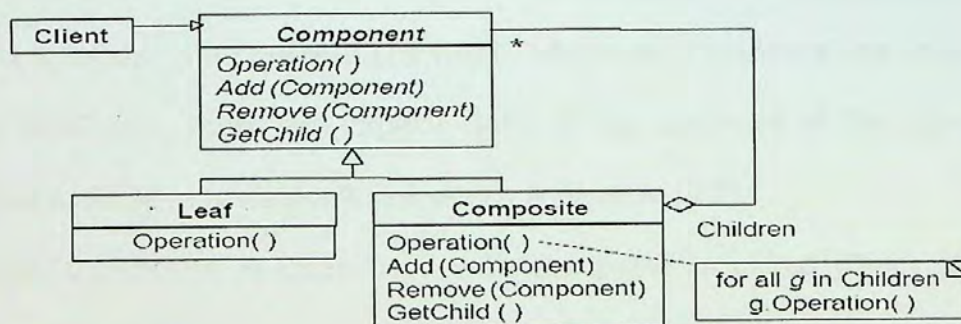
##### 1.2.2. Subtopic 1.2.2

#### 2. Topic 2



A simple implementation could define classes for simple elements, such as text and graphics, plus additional classes for subtopics as containers of simple elements. But in that case, the code using these classes would have to treat simple and container objects differently, which would be inefficient from the design point of view. Instead of that, the Composite pattern shows how to use recursive composition so that clients don't have to make this distinction. Moreover, using Composite makes it easier to achieve any desired depth of subtopics nesting [23].

**Structure.** The key to the Composite pattern is an abstract class that represents *both* primitive elements (concepts, text, and graphics) and their containers (subtopics). Figure (3-5) shows the general structure of the Composite pattern [23].



**Figure (3-5): Structure of the Composite Pattern [23]**

#### Participants and Collaborations.

- **Abstract class:** It corresponds to the contents to be presented during the lesson. It is responsible for the following:
  - Providing common interface to all objects in the composition, both simple (concepts, text, and graphics) and complex (topics and subtopics), like the *Add* and *Remove* functions illustrated in figure (3-5).
  - Defining the interface for accessing the child components of a complex object (a topic or a subtopic), e.g. the functions *Operation* and *GetChild*.
- **Leaf Class:** A concrete class that corresponds to simple contents of a lesson in our example). It represents leaf objects in the composition and has no children. It defines behavior for primitive objects in the composition.
- **Composite Class:** A concrete class that corresponds to topics and subtopics.

Both of the concrete classes are derived from *Component*. It implements child-related functions from the *Component* interface, stores child components, and defines behavior for components having children.

- **Client Object:** (e.g., a lesson presentation planner) interacts with any object in the composition through the common interface provided by the *Component* class. The client's requests are handled directly by *Leaf* recipients, and are usually forwarded to child components by *Composite* recipients.

**Known Uses.** The Composite pattern is used explicitly in the GET-BITS framework's lesson presentation planner and remedial actions planner (which is a part of the student's knowledge examination and assessment). The GET-BITS based FLUTE system, developed as an ITS for teaching formal languages, is a concrete ITS within which such planners are used. In the design of Eon tools and SimQuest, there are implicit signs of the presence of the Composite pattern, although the authors of these tools don't talk about it directly [23].

**Related Patterns.** Composite is often used with the patterns called Chain of Responsibility, Iterator, and Decorator [23].

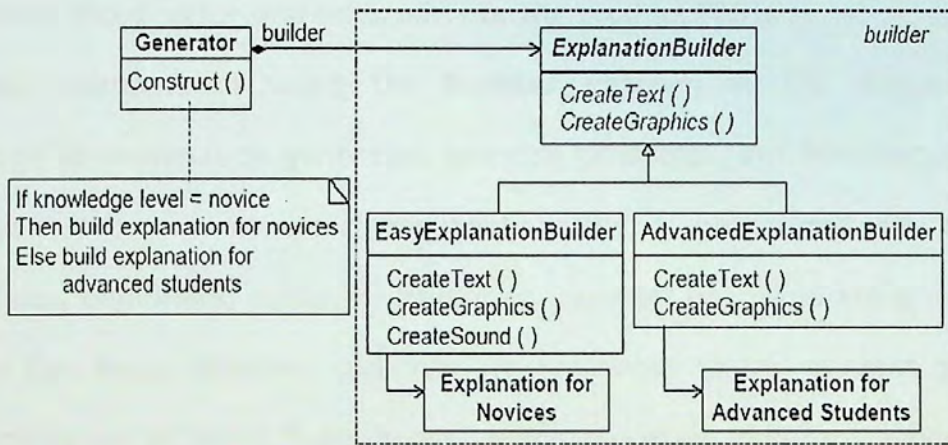
### 3.3.3 Builder Pattern

**Classification and Intent.** Builder is a creational pattern. Its intent is to help separate the construction of a complex object from its representation in order to make it possible to create different representations by the same construction process [23].

**Motivation.** Suppose an ITS designer wants to develop an explanation generator that can generate explanations for different students. In general, current level of mastering the subject of the ITS is different for different students at any given moment. That fact is reflected in the student model of each student. Novice students should get more general and easy explanations, while the ITS should generate more complex and detailed explanations to more advanced students. The problem is that the number of possible explanations of the same topic or process is open ended. It should be anticipated that during the system maintenance another set of knowledge levels can be introduced in order to describe the student model more accurately. The explanation generator should not be modified each time another set of knowledge levels is

introduced. On the contrary, it should be easy to add a new knowledge level easily [23].

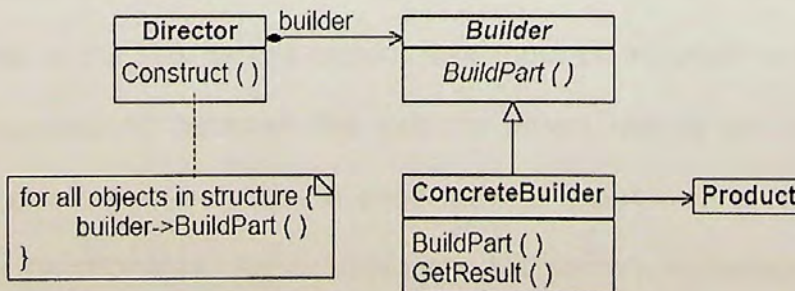
Using the **Builder Pattern** provides a solution: the explanation generator can be configured with an ExplanationBuilder, an object that converts a specific knowledge level from the student model to an appropriate type of explanation. Figure (3-6) illustrates this idea:



**Figure (3-6): Using the Builder Pattern in Designing Explanation Generator [23]**

Whenever the student requires an explanation, the explanation generator passes the request to the ExplanationBuilder object according to the student's knowledge level. Specialized explanation Builders, like EasyExplanationBuilder or AdvancedExplanationBuilder, are responsible for carrying out the request. Note that their concrete implementations of the functions like CreateText and CreateGraphics provide polymorphism in generating explanations [23].

**Structure.** Figure (3-7) shows the general structure of the **Builder Pattern**:



**Figure (3-7): General Structure of the Builder Pattern [23]**

### Participants and Collaborations.

The abstract class Builder (e.g., ExplanationBuilder) provides interface for creating parts of the Product object (e.g., text and graphics of the explanation). ConcreteBuilder implements that interface. The ITS designer configures the Director object (explanation generator) with the desired Builder object (types of explanation, according to the student's knowledge level). Director notifies the Builder each time a part of the product should be built. The Builder handles the requests from the director and adds parts to the product [23].

**Known Uses.** Examples of using the **Builder Pattern** in ITS design include different generators, such as explanation generator, exercise generator, and hint generator. In GET-BITS, explanation generator can construct explanations for a predefined set of users, which is configurable (e.g., beginners, midlevel, advanced, experts) problems are generated in much the same way. In Eon tools, different contents are presented to the student during the teaching process depending on different Topic levels, which represent different aspects or uses for the topic (e.g., introduction, summary, teach, test, beginning, difficult)[23].

**Related Patterns.** Builder goes together well with the Composite pattern, and is similar to the Abstract Factory pattern. In fact, the products that Builder constructs are often Composites [23].

### 3.4 A Pattern Language for Architectures of Intelligent Tutors

PLAIT, is a specific pattern language for architectures of intelligent tutors. It describes some patterns that exist in ITS architectures. The PLAIT pattern language is the first attempt to define a specific pattern language in the domain of ITSs Architectures [24].

#### 3.4.1 Inserted Layer

This is the basic pattern in PLAIT. *Inserted Layer* is used in ITS architectures whenever a new functionality is needed in the ITS, or the architecture must be adapted to a new requirement, or some translation (negotiation) between the existing layers has to be performed. In order to better decouple the existing layers, it is a good idea to insert a new layer to implement and encapsulate the new functionality. An example on this pattern is introducing the believability layer in ITSs [24]. This pattern is illustrated in figure (3-8)a:

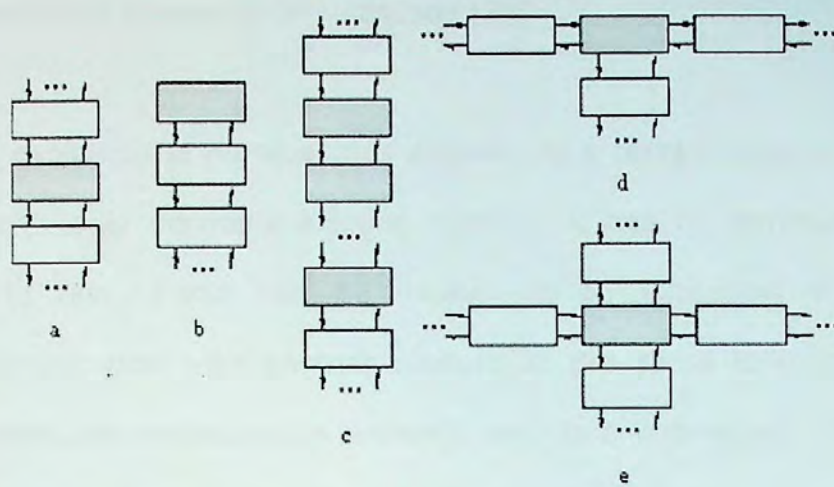


Figure (3-8)

- a) The Inserted Layer Pattern b) The Top Pattern c) The Cascade Pattern d) The T-join Pattern e) The Cross Pattern[24]

### 3.4.2 Top

The name *Top* reflects the idea of putting a new layer on top of other layers as shown in figure (3-8)b. It is typically used when it is necessary to hide the complexity of the layered architecture from the outside world, as when the user or another system should have a specific or a well-defined "interface" for communication with a layered ITS. For example, in the SQL Tutor the Constraint-Based Modeling (CBM) layer is put on top of the pedagogical module and the user interface in order to overcome computational intractability of learner modeling [24].

### 3.4.3 Cascade

A cascade of new, distinct layers, hence the name *Cascade* as shown in figure (3-8)c. This pattern is good to use when a new functionality is needed in a layered intelligent tutor, and it can be represented as a strict series of simpler modules. Also, sometimes a pipeline-style agent communication should be provided in an agent-based ITS, or a complex existing module should be broken into a sequence of simple functions and there must be a strict order of function calls. In such cases, the solution is to represent each distinct functionality/agent/function as an individual module with strictly specified input and output, and make a cascade of such modules according to the observed strict processing sequence. An example is Knowledge Awareness (KA) agent of the Sharlok learning environment which has its history observer, KA generator, KA

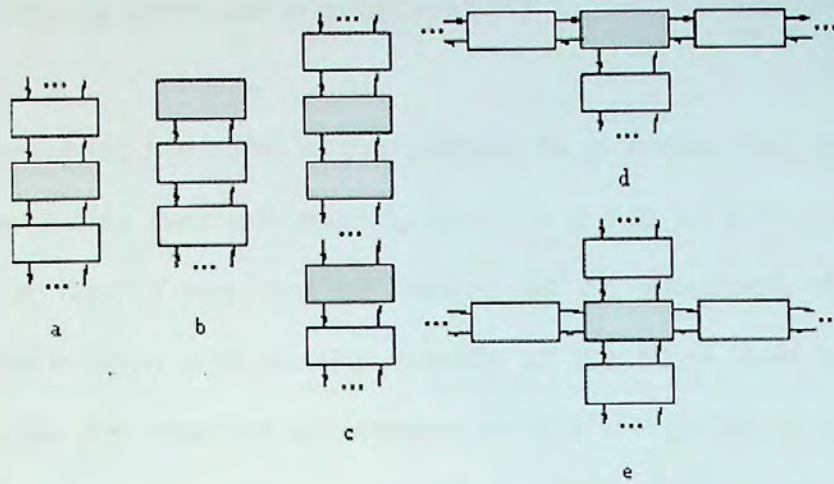


Figure (3-8)

- a) The Inserted Layer Pattern b) The Top Pattern c) The Cascade Pattern d) The T-join Pattern e) The Cross Pattern[24]

### 3.4.2 Top

The name *Top* reflects the idea of putting a new layer on top of other layers as shown in figure (3-8)b. It is typically used when it is necessary to hide the complexity of the layered architecture from the outside world, as when the user or another system should have a specific or a well-defined "interface" for communication with a layered ITS. For example, in the SQL Tutor the Constraint-Based Modeling (CBM) layer is put on top of the pedagogical module and the user interface in order to overcome computational intractability of learner modeling [24].

### 3.4.3 Cascade

A cascade of new, distinct layers, hence the name *Cascade* as shown in figure (3-8)c. This pattern is good to use when a new functionality is needed in a layered intelligent tutor, and it can be represented as a strict series of simpler modules. Also, sometimes a pipeline-style agent communication should be provided in an agent-based ITS, or a complex existing module should be broken into a sequence of simple functions and there must be a strict order of function calls. In such cases, the solution is to represent each distinct functionality/agent/function as an individual module with strictly specified input and output, and make a cascade of such modules according to the observed strict processing sequence. An example is Knowledge Awareness (KA) agent of the Sharlok learning environment which has its history observer, KA generator, KA

filtering and KA monitoring connected in a cascade [24].

#### **3.4.4 T-join**

Figure (3-8)d itself explains the name of this pattern. At a certain layer, a new functionality is needed but it doesn't only decouple existing layers - it has to communicate with another, "lateral" module. In fact, T-join can be viewed as an extension of Inserted Layer to accommodate communication with another module at the same level of abstraction in the hierarchy. For example, decomposing the learner's input in a Web-based ITS may require a new layer - a parser - between the user interface and the translator [24].

#### **3.4.5 Cross**

This pattern has got its name from the fact that "vertical" and "horizontal" communication between some modules cross at a specific module as shown in figure (3-8)e. A new functionality is needed and it has to communicate with two distinct "lateral" modules. An example is the Vincent animated pedagogical agent; its action module is at the cross between the Mind module, the Body module, Micro Learning Environment, and the user interface [24].

## Chapter 4

# The Proposed Pattern Language for Intelligent Tutoring Systems (PLITS)

In this research, we first identified the functional requirements of ITS which will be summarized in table (9) and then tried to discover these features in a number of real ITSs that are broadly used. If these features were indeed found in existing ITSs, then these features were considered widely adopted and applicable and were therefore included in our pattern language for intelligent tutoring systems. The set of ITSs that we used to mine the patterns is shown in the following table:

**Table (4): ITSs That Were Searched for Patterns**

Name	Domain
SQLT-Web [5]	Database Learning(SQL)
Cyberphysique [25]	Physics
Web Passive Voice Tutor-Web PVT[17]	Language Learning
Verb Expert[20]	Language Learning
CAPIT: An Intelligent Tutoring System for Capitalization and Punctuation[19]	Language Learning
Intelligent Language Tutoring System for Grammar Practice[15]	Language Learning
Learning Greek with an Adaptive and Intelligent Hypermedia System[26]	Language Learning
An Interactive Course Support System for Greek[18]	Language Learning

When we were formulating **PLITS** we wanted to provide a road map for any developer or designer to follow when faced by the problem of implementing an ITS. Thus we followed a comprehensive approach to cover all aspects related to ITS implementation.



#### 4.1 PLITS Pattern Categories

All the patterns included in **PLITS** belong to one of the following pattern categories:

##### 1. Access patterns

Access Patterns are concerned with the ways that user may access the various resources of Intelligent Tutoring Systems [27].

##### 2. Instructional patterns

Instructional Patterns are concerned with the various tasks that tutors perform in order to create and edit courses and learning resources [27].

**Both Instructional patterns and Access patterns** are used in the Learning management Systems domain. However, we found out that they can be useful in ITS implementation if they were modified to suit the ITS needs.

##### 3. Design Patterns

Design Patterns can be divided into the following subcategories:

###### a. Creational Patterns

Creational Patterns abstract the instantiation process. They help make a system independent of how its objects are created, composed, and represented [7].

###### b. Structural Patterns

Structural patterns are concerned with how classes and objects are composed to form larger structures. Structural class patterns use inheritance to compose interfaces or implementations [7].

###### c. Behavioral Patterns

Behavioral patterns are concerned with algorithms and the assignment of responsibilities between objects. Behavioral patterns describe not just patterns of objects or classes but also the patterns of communication between them [7].

##### 4. Adaptive Patterns

Adaptive instruction can be defined as real-time modification of the instructional curriculum, learning environment, or training regimen to suit different student characteristics [28]. Adaptive patterns are patterns that is used in Adaptive learning systems. In our search for patterns that

can be useful in ITS implementation we found that there is a number of adaptive learning patterns that can be of great use to the designers and developers of ITSs if they were modified to suit the ITS needs.

### 5. Pedagogical Patterns

Pedagogical Patterns are concerned with patterns that represent the best practices for teaching and education. The intent of pedagogical patterns is to capture the essence of the practice in a compact form that can be easily communicated to those who need the knowledge and to present this information in a coherent and accessible form [29]. In our search for patterns that can be useful in ITS implementation we found that there is a number of adaptive learning patterns that can be of great use to the designers and developers of ITSs if they were modified to suit the ITS needs.

### 6. Interaction Patterns:

Interaction Patterns are focused on solutions to problems end-users have when interacting with systems. The patterns take an end-user perspective which leads to a format where usability is the essential design quality [30].

It is important to note that some of the discovered patterns were based on existing patterns; however, we modified some of these patterns in order to suit the needs of ITS implementation. All modified patterns have new names that begin with the word "New".

### 4.2 PLITS Pattern Template

Documenting a design pattern is the first step to build a pattern language. We used two pattern description formats when describing the patterns in **PLITS**; namely:

- **GOF pattern template to describe the design patterns.** The GOF format is very complete and provides straightforward guidelines for implementing the patterns into software. We used it in describing design patterns because we felt that this category of patterns requires more implementation details rather than generic solutions [30].

- **A variant of the Alexandrian template.** The Alexandrian format is a rather abstract way of describing patterns, as it does not delve into implementation details, but rather expresses a generic solution. The reason for choosing the Alexandrian rather than the GOF format was that all the pattern categories other than the design patterns do not contain many implementation details, but are rather generic and abstract and can be implemented in several different ways. The same practice is used in the hypermedia patterns as well as the Human Computer Interface patterns [30].

#### 4.2.1 GOF Pattern Template [7]

This template consists of the following elements:

- **Name**
- **Classification**

The classification describes the family of patterns that the described pattern belongs to. This classification differs according to pattern purpose and scope.

- **Intent**

The intent is a short statement that answers the following questions: What does the design pattern do? What is its rationale and intent? What particular design or issue does it address?

- **Motivation**

A scenario that illustrates a design problem and how the class and object structures in the pattern solve the problem. The scenario will help in understanding the more abstract description of the pattern that follows.

- **Applicability**

What are the situations in which the design pattern can be applied? What are examples of poor designs that the pattern can address? How can you recognize these situations?

- **Structure**

The structure is a graphical representation of the classes in the pattern.

- **Participants**

The participants are classes and/or objects participating in the design pattern and their responsibilities.

- **Consequences**

How does the pattern support its objectives? What are the trade offs and results of using the pattern? What aspect of system structure does it let you vary independently?

- **Implementation**

What pitfalls, hints or techniques should you be aware of when implementing the pattern? Are there language specific issues?

- **Sample code**

Code fragments that illustrate how you might implement the pattern.

- **Known uses**

The Known uses are examples of the pattern in real ITS.

- **Related Patterns**

What design patterns are closely related to this one? What are the important differences? With which other patterns should this one be used?

- **Anti-patterns (optional)**

Solutions that are inferior or do not work in this context. The reason for their rejection should be explained. The anti-patterns may be valid solutions in other contexts, or may never be valid.

- **References**

The references are the acknowledgements of those who developed or inspired the pattern.

#### 4.2.2 Alexandrian Pattern Template [30]

This template is a variant of the Alexandrian format that contains the following elements:

- **Name**
- **Classification**

The classification describes the family of patterns that the described pattern belongs to. This classification differs according to pattern purpose and scope.

- **Problem**

The problem is a brief description of the design problem at hand.

- **Usability Principle** (used only with interaction patterns)

Interaction patterns usually use a 'principle' on which the solutions are based. The following list

of usability principles is used grouped according to user problem categories:

- o **Visibility:** User guidance, Grouping, Incremental Revealing
- o **Affordance:** Metaphors
- o **Natural Mapping:** Compatibility
- o **Constraints:** Minimizing actions, Self-explanation
- o **Conceptual Models:** Significance of codes, Compatibility, Adaptability
- o **Feedback:** Immediate Feedback, Legibility
- o **Safety:** Error Prevention, Error Correction, Forgiveness
- o **Flexibility:** Explicit control

- **Motivation**

The motivation is an explanation of the origins of the problem, probably with an example for better communicating it. It may also contain the context of the particular problem if it is necessary in order to make it more comprehensible.

- **Solution**

A description of the solution proposed by this pattern that addresses the problem and motivation stated earlier.

- **Known uses**

The Known uses are examples of the pattern in real ITS.

- **Related Patterns**

The related patterns are the other patterns that are related to this one in some way.

- **References**

The references are the acknowledgements of those who developed or inspired the pattern.

### 4.3 Building an Intelligent Tutoring System

Before discussing how to build an ITS using our proposed pattern language we must point out that **PLITS** contains 19 patterns as listed in Table (9). A Typical ITS is built in the following phases:

### **4.3.1 Phase One: Building the Domain Module**

The following patterns are needed in building the domain module.

#### **4.3.1.1 Pattern 1: Early Bird [31]**

Organize the course so that the most important topics are taught first. Teach the most important material, the "big ideas," first [31].

#### **Classification**

Pedagogical Patterns

#### **Problem**

The typical course has many important topics. Many times they are interrelated. It is difficult to decide how to order topics so that students will appreciate the "big ideas" in the course. If the tutor delayed important topics until late in the course, spending much time on preliminaries, students may get the wrong idea about relative importance. The tutor will also not be able to reinforce the big ideas frequently through follow up exercises and discussions [31].

#### **Motivation**

Students need to know where they are headed. They need to see that detail presented early in the course will relate to important ideas. Students need to know what the few big ideas from each course are. They need to be able to separate the key concepts from the detail that support them. Students often remember best what they learn first. Important ideas can be introduced early, even if they can't get complete treatment immediately [31].

#### **Solution**

The tutor must first identify the most important ideas in the course. These ideas become the fundamental organizational principle of the course. Then he must introduce these big ideas and their relationships at the beginning of the course and return to them repeatedly throughout the course [31].

Implementation is difficult. Often only simple aspects of an important idea can be introduced early. Sometimes it is enough to give important terms and general ideas. Some "big" ideas are thought of as advanced. It is difficult to introduce some of these early. Hard thought and preparation are needed in curriculum design. Sometimes a really big, but difficult, concept can be introduced incompletely. Then as other material that relates to it is covered, the relationship to the big idea is carefully explored [31].

However, if the tutor can not introduce the big idea at the beginning, then he has to make certain that nothing that is done early in the course material is inconsistent with the big idea or impedes its easy learning. The tutor needs to be able to analyze deeply what are the consequences of developing material in a particular order [31].

### Known Uses

1. **CAPIT [19]**: An ITS that teaches the rules of English capitalization and punctuation. CAPIT used both an experienced teacher feedback and text books in order to determine big ideas that should be introduced first.
2. **Learning Greek with an Adaptive and Intelligent Hypermedia System[26]**:An ITS for English learners of Greek .This ITS was supported by the Greek Ministry of Education that provided it with both an experienced teacher feedback and text books in order to determine big ideas that should be introduced first .
3. **Cyberphysique [25]**: An intelligent tutoring system based on the World Wide Web that teaches physics. This ITS was supported by the Canadian Ministry of Education that provided it with experienced teachers feedback in order to determine big ideas that should be introduced first .

### Related Patterns

Whole Part

Course Creation and Customization

### References

The **Early Bird Pattern** is one of the fourteen pedagogical patterns [31].

### 4.3.1.2 Pattern 2: The Whole Part Pattern

#### Classification

Design Patterns

#### Intent

The **Whole Part Pattern** helps with the aggregation of components that together form a semantic unit. An aggregate component, the Whole, encapsulates its constituent components, the Parts, organizes their collaboration, and provides a common interface to its functionality [12].

#### Motivation

In almost every software system there exist objects that are composed of other objects. Such aggregate objects do not represent loosely coupled set of components. Instead, they form units that are more than just a mere collection of their parts [12].

**Typical situations in which the Whole Part Pattern can be applied in Domain Module in ITS design:**

- Composing Topics
- Composing Lessons
- Composing Curricula

#### Applicability

**Use the Whole Part Pattern When [12]:**

- You want to introduce a component that encapsulates smaller objects
- You want to represent part whole hierarchies of objects.

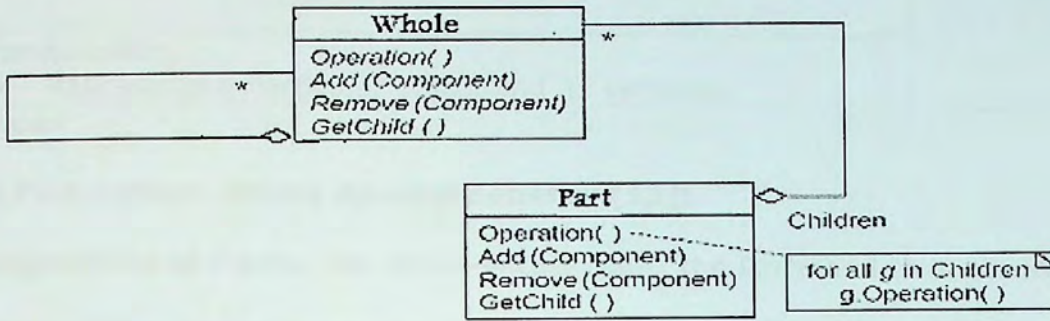
**Before discussing the usage of the Whole Part Pattern in ITS it is important to indicate that:**

- Teaching Material is composed of a number of topics.
- Each Topic is composed of a number of learn items.
- Each Topic has a number of exercises related to it.
- Each Learn Item has a number of examples related to it.



**Structure**

Figure (4-1) shows the structure of the **Whole Part Pattern**.



**Figure (4-1): Whole Part Pattern Structure**

**Participants**

The Whole Part Pattern introduces two types of Participants:

- Whole Object
- Part Object

These participants can be described in more detail as follows:

- A **Whole object** that represents an aggregation of smaller objects, which we call **Parts**. It forms a semantic grouping of its Parts in that it coordinates and organizes their collaboration. For this purpose, the Whole uses the functionality of Part objects for implementing services [12].

Some methods of the Whole may be just placeholders for specific Part services. When such a method is invoked the Whole only calls the relevant Part service and returns the result to the client [12].

**Table (5): Whole Class Responsibility and Collaborators [12]**

Class	Collaborators
Whole	<ul style="list-style-type: none"> <li>• Part</li> </ul>
<b>Responsibility</b> <ul style="list-style-type: none"> <li>• Aggregates several smaller objects.</li> <li>• Provides services built on top of Part objects.</li> </ul>	

**Table (6): Part Class Responsibility and Collaborators [12]**

Class	Collaborators
Part	
<b>Responsibility</b> <ul style="list-style-type: none"> <li>• Represents a Particular object and its services.</li> </ul>	

**Consequences****The Whole Part Pattern Offers Several Benefits [12]:**

- **Changeability of Parts** .The whole encapsulates the Parts and thus conceals them from its clients. This makes it possible to modify the internal structure of the Whole without any impact on clients. Part implementations may even be completely changed without any need to modify other Parts or clients.
- **Separation of Concerns**. A Whole Part structure supports the separation of concerns. Each concern is implemented by a separate Part. It therefore becomes easier to implement complex strategies by composing them from simpler services than to implement them as monolithic units.
- **Reusability**. The **Whole Part Pattern** supports two aspects of reusability:
  - Parts of a Whole can be reused in other aggregate objects.
  - The encapsulation of Parts within a Whole prevents a client from scattering the use of Part objects all over its source code this supports the reusability of Wholes.

**Implementation [12]****To implement a Whole Part structure, apply the following steps:**

1. **Design the Public Interface of the Whole**. Analyze the functionality the Whole must offer to its clients. Only consider the client's viewpoint in this step. Think of the Whole as an atomic component that is not structured into Parts, and compile a list of methods that together comprise the public interface of the Whole.

2. **Separate the Whole into Parts**.

Partitioning into Parts is driven by the services the Whole provides to its clients, freeing you from the requirement to implement glue code.

3. Partition the Whole's services into smaller collaborating services and map these collaborating services to separate Parts.

4. **Define the Public Interface of the Parts.**

5. **Implement the Parts.**

If the Parts are Whole Part structures themselves, design them recursively starting with step 1. If not, reuse existing Parts from a library, or just implement them if their implementation is straightforward and further decomposition is not necessary.

6. **Implement the Whole.** Implement the Whole's services based on the structure developed in the preceding steps. Implement services that depend on Part objects by invoking their services from the Whole. Implement those services that do not depend on a Part object in this step.

When implementing the Whole, you need to take any given constraints into account, such as cardinality properties. We also need to manage the life cycle of Parts. Since a Part lives and must therefore die with its Whole, the Whole must be responsible for creating and deleting the Part.

#### **Sample Code**

Please refer to appendix D for sample source code.

#### **Known Uses**

1. **GET-BITS [32]** : GET-BITS is an object-oriented model of intelligent tutoring systems .GET-BITS Uses a variant of the **Whole Part Pattern** in the framework's lesson presentation planner and remedial actions planner (which is a part of the student's knowledge examination and assessment).
2. **Eon Tools [33]**: "Eon" is the name for a suite of authoring tools for building intelligent tutoring systems. Eon includes tools for authoring all aspects of intelligent tutors, including the learning environment, the domain knowledge, the teaching strategies, and the student model. Eon Tools uses the **Whole Part Pattern** in its lesson presentation planner to represent the teaching material and its constituent parts.

3. **SimQuest [34]**: SimQuest is an intelligent biomedical simulator for medical training utilizing the expertise of professionals in the fields of 3D graphics programming, clinical training, game and simulation design, medical illustration, human factors and mechanical engineering. SimQuest focuses on the use of advanced simulation and gaming technologies for medical training. SimQuest uses the **Whole Part Pattern** in its lesson presentation planner to represent the teaching material and its constituent parts.

### Related Patterns

Early Bird

Course Creation and Customization

Singleton

New Flyweight

Builder

New Strategy

New Adapter

New Student Model Initialization

Template Method

### References

The **Whole Part Pattern** is one of the POSA (Pattern Oriented Software Architecture) patterns [12].

#### 4.3.1.3 Pattern 3: Singleton

##### Classification

Creational Pattern

##### Intent

Ensure a class only has one instance, and provide a global point of access to it [7].

##### Motivation

It is important for some classes to have exactly one instance. How can we ensure that a class has only one instance and that the instance is easily accessible? A global variable makes an object accessible, but it does not keep us from instantiating multiple objects [7].

A better solution is to make the class itself responsible for keeping track of its sole instance. The class can ensure that no other instance can be created by intercepting requests to create new objects, and it can provide a way to access the instance [7].

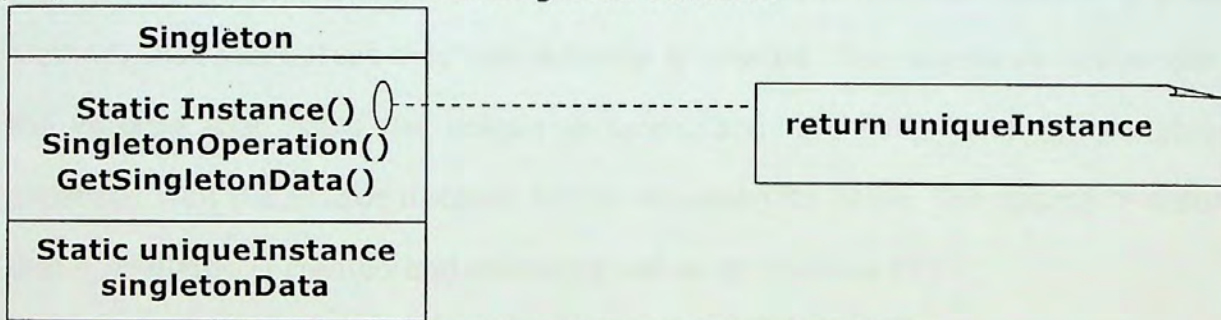
**Applicability**

**Use Singleton Pattern when [7]:**

- There must be exactly one instance of a class, and it must be accessible to clients from a well known access point.
- When the sole instance should be extensible by sub classing, and clients should be able to use an extended instance without modifying their code.

**Structure**

Figure (4-2) represents the structure of the **Singleton Pattern**.



**Figure (4-2): Singleton Pattern Structure [7]**

**Participants [7]**

- **Singleton**
  - Defines an abstract operation that lets clients access its unique instance. Instance is a class operation.
  - May be responsible for creating its own unique instance.

**Consequences**

**The Singleton Pattern has several benefits [7]:**

1. **Controlled access to sole instance.** Because the Singleton class encapsulates its sole instance, it can have strict control over how and when clients access it.
2. **Reduced name space.** The **Singleton Pattern** is an improvement over global variables. It avoids polluting the name space with global variable that store sole instances.

3. **Permits a variable number of instances.** The pattern makes it easy to change your mind and allow more than one instance of the Singleton class. Moreover, the same approach can be used to control the number of instances that the application uses. Only the operation that grants access to the Singleton instance needs to change.

### Implementation

Here are implementation issues to consider when using the Singleton Pattern:

- **Ensuring a unique instance.** The **Singleton Pattern** makes the sole instance a normal instance of a class, but that class is written so that only one instance can ever be created. A common way to do this is to hide the operation that creates the instance behind a class operation (that is, either a static member function or a class method) that guarantees only one instance is created. This operation has access to the variable that holds the unique instance, and it ensures that the variable is initialized with the unique instance before returning its value. The approach ensures that a Singleton is created and initialized before its first use [7].

The Singleton Pattern can be implemented using Java through the following steps:

1. Create the Singleton class with a private constructor, so that no one outside the class can create an object.
2. Create a member field that is private (to be non directly accessible to outside code) and static that has the same type as the class.
3. Create a public method that lets clients get the unique instance. In this method it should be checked if the instance is created or not. If it is created it will be returned if not it will be created then returned.

It is important to note that in the **Singleton Pattern** there must not be only one instance ;there can be a predefined number of instances however there is usually a constraint on the maximum number of instances that can be used.

### Sample code

Please refer to appendix D for sample source code.

### Related Patterns

Whole Part

Builder

Observer

### References

The **Singleton Pattern** is one of the Gang of Four patterns [7].

#### 4.3.1.4 Pattern 4: New Flyweight

##### Classification

Structural Patterns

##### Intent

Use sharing to support large numbers of fine grained objects efficiently. The **New Flyweight Pattern** describes how to share objects to allow their use at fine granularities without prohibitive cost. Flyweights model concepts or entities that are normally too plentiful to represent with objects.

##### Motivation

Some applications could benefit from using objects throughout their design, but a naïve implementation would be prohibitively expensive [7].

A Flyweight is a shared object that can be used in multiple contexts simultaneously. The Flyweight acts as an independent object in each context-it is indistinguishable from an instance of the object that's not shared. Flyweights cannot make assumptions about the context in which they operate [7].

**The key concept here is the distinction between intrinsic and extrinsic state:**

- **Intrinsic state:** Is stored in the Flyweight; it consists of information that's independent of the Flyweight's context, thereby making it sharable.
- **Extrinsic state:** Depends on and varies with the flyweight's context and therefore can't

be shared. Client objects are responsible for passing extrinsic state to the flyweight when it needs it [7].

**The New Flyweight Pattern is needed in the context of ITS in two issues:**

- **Teaching Material Instantiation:** Since all the students need to view the teaching material; the **New Flyweight Pattern** allows for the instantiation of this shared object to avoid duplication and prohibitive cost. Thus allowing for the use of the shared object (Teaching Material) in multiple contexts by multiple students simultaneously.
- **Arabic Alphabetic Letters and Shapes Instantiation:** This is an ILTS specific issue. Thus allowing for the use of the shared object (Alphabetic Letters and Shapes) in multiple contexts by multiple students simultaneously.

**Applicability**

The **New Flyweight Pattern's** effectiveness depends heavily on how and where it's used.

**Apply the New Flyweight Pattern when all of the following are true:**

- An application uses a large number of objects.
- Storage costs are high because of the sheer quantity of objects.
- Most object states are made extrinsic.
- Many groups of objects may be replaced by relatively few shared objects once extrinsic is removed.
- The application doesn't depend on object identity. Since Flyweight objects may be shared, identity tests will return true for conceptually distinct objects.

**Typical situations in which the New Flyweight Pattern can be applied in ITS design:**

- Instantiating Teaching material.
- Alphabetic Letters and Shapes Instantiation.



Structure

Figure (4-3) shows the structure of the **New Flyweight Pattern**.

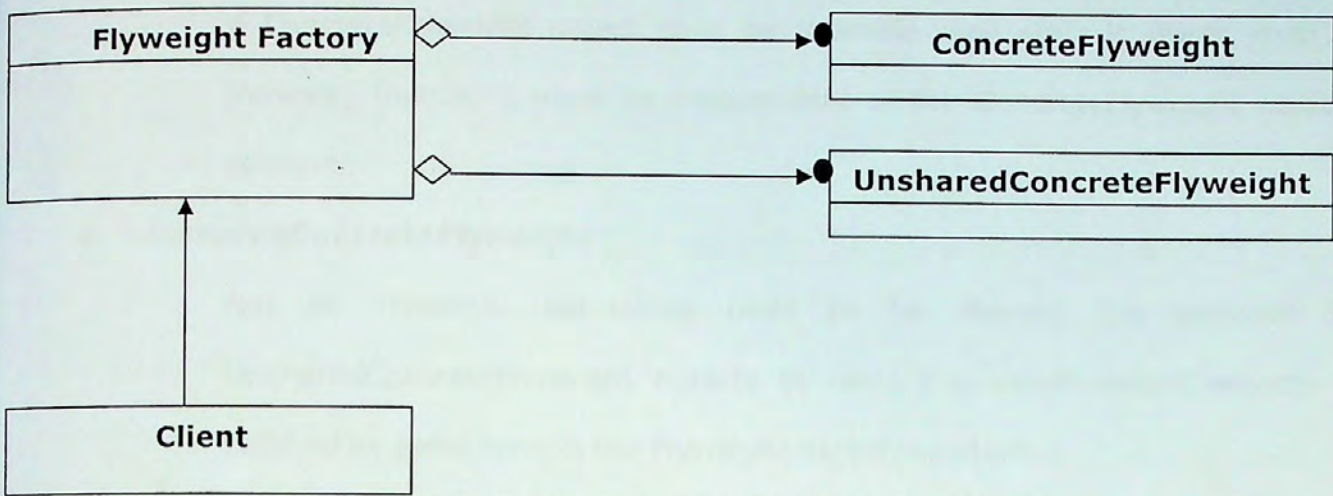


Figure (4-3): New Flyweight Pattern Structure

The reason for naming this pattern New Flyweight was that it is a variant of the **Flyweight Pattern** whose structure is represented in figure (4-4):

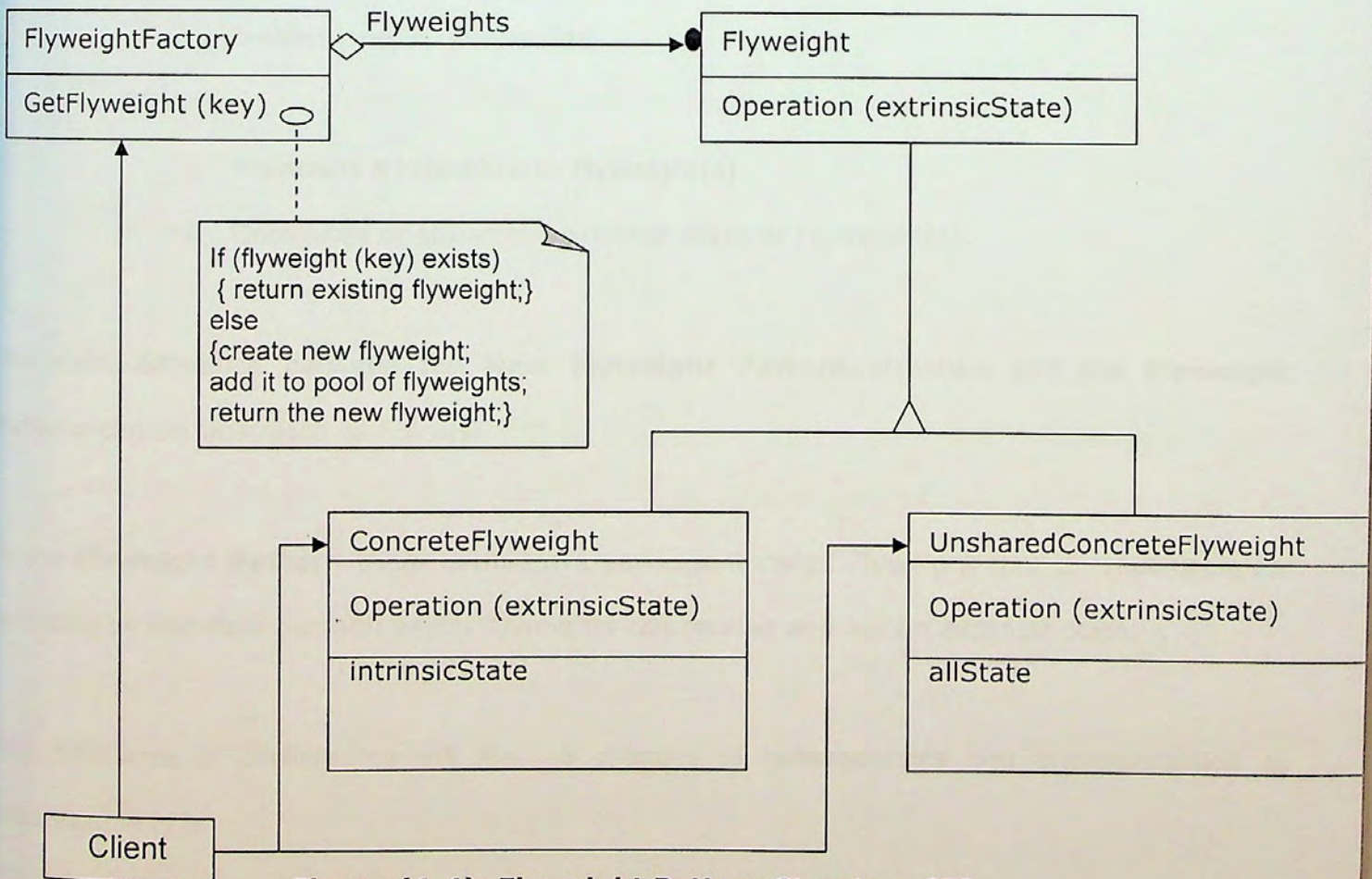


Figure (4-4): Flyweight Pattern Structure [7]

## Participants

- **ConcreteFlyweight**
  - A ConcreteFlyweight object must be sharable. Any state it stores must be intrinsic; that is, it must be independent of the ConcreteFlyweight object's context.
- **UnsharedConcreteFlyweight**
  - Not all Flyweight subclasses need to be shared. It's common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the Flyweight object structure.
- **Flyweight Factory**
  - Creates and manages Flyweight objects.
  - Ensures that Flyweights are shared properly. When a client requests a Flyweight, the FlyweightFactory object supplies an existing instance or creates one, if none exists.
- **Client**
  - Maintains a reference to Flyweight(s).
  - Computes or stores the extrinsic state of Flyweight(s).

The main difference between the **New Flyweight Pattern** structure and the **Flyweight Pattern** can be illustrated as follows:

In the **Flyweight Pattern** there is an extra participant called Flyweight that is responsible for declaring an interface through which flyweights can receive and act on extrinsic state.

This difference in participants will lead to changes in consequences and implementation as illustrated in [7].

## Consequences

Flyweights may introduce run time costs associated with transferring, finding, and/or computing extrinsic state, especially if it was formerly stored as intrinsic state. However, such costs are offset by space savings, which increase as more Flyweights are shared [7].

### Storage savings are functions of several factors [7]:

- The reduction in the total number of instances that comes from sharing.
- The amount of intrinsic state per object.
- Whether extrinsic state is computed or stored.

The more Flyweights are shared, the greater the storage savings. The savings increase with the amount of shared state. The greatest savings occur when the objects use substantial quantities of both intrinsic and extrinsic state, and the extrinsic state can be computed rather than stored. Then you save on storage in two ways: sharing reduces the cost of intrinsic state, and you trade extrinsic state for computation time [7].

## Implementation

Consider the following issues when implementing the New Flyweight Pattern in its general form:

- **Removing extrinsic state.** The pattern's applicability is determined largely by how easy it is to identify extrinsic state and remove it from shared objects. Removing extrinsic state won't help reduce storage costs if there are as many different kinds of extrinsic state as there are objects before sharing. Ideally, extrinsic state can be computed from a separate object structure, one with far smaller storage requirements [7].
- **Managing shared objects.** Because objects are shared, clients shouldn't instantiate them directly. FlyweightFactory lets clients locate a particular flyweight. FlyweightFactory objects often use an associative store to let clients look up flyweights of interest [7].

Sharability also implies some form of reference counting or garbage collection to reclaim a Flyweight's storage when it is no longer needed [7].

## Sample Code

Please refer to appendix D for sample source code.

## Related Patterns

Whole Part

Builder

## References

This pattern was based on the **Flyweight Pattern** [7], however, through our search for patterns in the field of ITS we discovered that this pattern can be modified to suit the needs of ITS developers and designers.

### 4.3.1.5 Pattern 5: Builder

#### Classification

Creational Patterns

#### Intent

The **Builder Pattern** helps to separate the construction of a complex object from its representation. Such a separation makes it possible to create different representations by the same construction process [7].

#### Motivation

The **Builder Pattern** is responsible for separating the construction algorithm from the representation in order to allow different representations for the object that is constructed [7].

#### Applicability

The **Builder Pattern** is useful whenever the algorithm for creating complex object should be independent of the parts that make up the object and how they are assembled. It is also useful when different representations are needed for the object that's constructed [7].

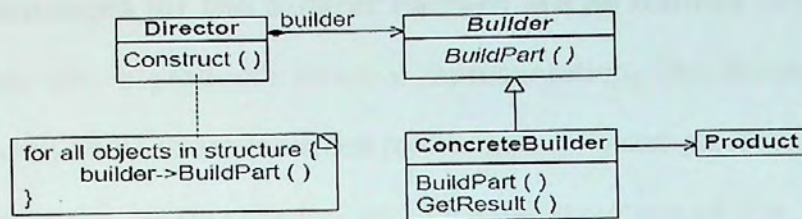
#### Use the Builder Pattern when [7]:

- The algorithm for creating a complex object should be independent of the parts that make up the object and how they're assembled.
- The construction process must allow different representations for the object that's constructed.

In ITS implementation the **Builder Pattern** is needed to construct and load the teaching material that acts as the nucleus of the domain module.

## Structure

Figure (4-5) shows the general structure of the **Builder Pattern**.



**Figure (4-5): Builder Pattern Structure [7]**

In any ITS there is an aggregation relationship that is illustrated as follows:

- Teaching Material is composed of a number of topics.
- Each Topic is composed of a number of learn items.
- Each Topic has a number of exercises related to it.
- Each Learn Item has a number of examples related to it.

## Participants [7]

- **Builder**
  - Specifies an abstract interface for creating parts of a Product object.
- **ConcreteBuilder**
  - Constructs and assembles parts of the product by implementing the Builder interface.
  - Defines and keeps track of the representation it creates.
  - Provides an interface for retrieving the product.
- **Director**
  - Constructs an object using the Builder interface.
- **Product**
  - Represents the complex object under construction. ConcreteBuilder builds the product's internal representation and defines the process by which it's assembled.
  - Includes classes that define the constituent parts, including interfaces for

assembling parts into the final result.

## Consequences

The key consequences for the Builder Pattern are as follows [7]:

1. It lets you vary a product's internal representation. The Builder object provides the director with an abstract interface for constructing the product. The interface lets the builder hide the representation and internal structure of the product. It also hides how the product gets assembled. Because the product is constructed through an abstract interface, all you have to do to change the product's internal representation is define a new kind of Builder.
2. It isolates code for construction and representation. The **Builder Pattern** improves modularity by encapsulating the way a complex object is constructed and represented. Clients needn't know anything about the classes that define the product's internal structure; such classes don't appear in Builder's interface.

Each ConcreteBuilder contains all the code to create and assemble a particular kind of product. The code is written once; then different Directors can reuse it to build Product variants from the same set of parts.

3. It gives you finer control over the construction process. Unlike creational patterns that construct products in one shot, the **Builder Pattern** constructs the product step by step under the director's control. Only when the product is finished does the director retrieve it from the Builder. Hence the Builder interface reflects the process of constructing the product more than other creational patterns. This gives you finer control over the construction process and consequently the internal structure of the resulting product.

## Implementation

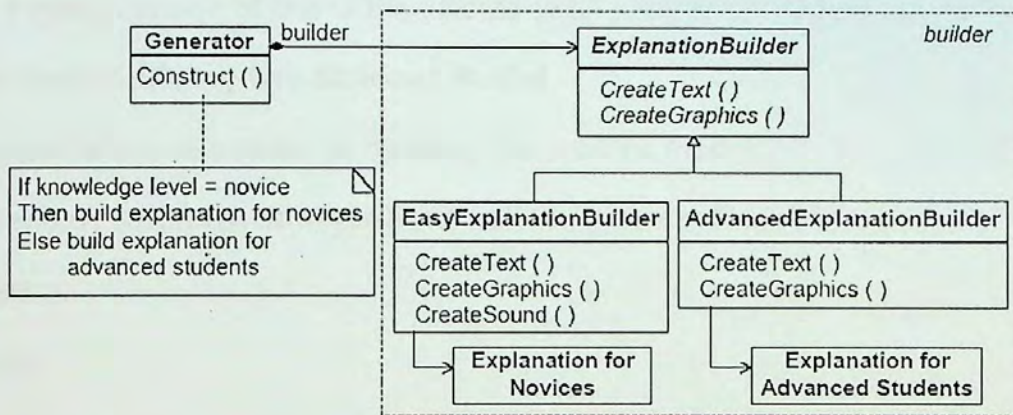
The most important issue to consider when implementing the **Builder Pattern** is that builders construct their products in step by step fashion. Therefore the builder class interface must be general enough to allow the construction of products for all kinds of concrete builders.

**Sample code**

please refer to appendix D for sample source code.

**Known Uses**

1. **GET-BITS [32]:** GET-BITS is an object-oriented model of intelligent tutoring systems .GET-BITS Uses **Builder Pattern** to develop an explanation generator that can generate different explanations for different students according to their current knowledge level. The structure of the **Builder Pattern** in Get-Bits is illustrated in figure (4-6):



**Figure (4-6): Using Builder Pattern in Implementing Explanation Generator in Get-Bits [32]**

2. **The Web Based German Tutor [21]:** A web based intelligent tutoring system for teaching German. The current knowledge level of the student determines the specificity of the feedback messages that he receives when he makes a mistake in one of the exercises. The more experienced the student is the less detailed feedback he gets. Tailoring feedback messages according to student level follows the pedagogical principle of guided discovery learning. Guided discovery takes the student along a continuum from heavily structured, tutor directed learning to a point where the tutor plays less of a role.
3. **An Interactive Course Support for Greek [18]:** An intelligent tutoring system for teaching Greek. It has four student stereotypes novice, beginner, intermediate and expert. The current knowledge level of the student determines the specificity of the

feedback messages that he receives when he makes a mistake in one of the exercises.

### Related Patterns

Whole Part

Singleton

New Flyweight

New Adapter

### References

The **Builder Pattern** is one of the GOF patterns [7].

#### 4.3.2 Phase Two: Building the Student Model

The following patterns are needed in building the student model.

##### 4.3.2.1 Pattern 6: Registration-Authentication-Access Control

#### Classification

Access Patterns

#### Problem

How can all the different students' access rights and privileges be effectively managed?

#### Motivation

ITSs are large, multi-user systems. Due to security, privacy and institutional policy reasons, students' access to the resources of the ITS must be restricted to authorized students only.

Additionally, user roles vary and can be divided into two major roles:

- **Teacher Roles:** Teachers access the ITS to add teaching material including new topics, learn items, examples and exercises.
- **Student Roles:** Students access the ITS in order to learn a certain course and benefit from the intelligence and adaptivity of the ITS.

Consequently, systems must assign specific rights to the various systems resources according to the role of each user.



**Solution**

Provide a standard registration mechanism for every user of the system. Users may register themselves through a web interface. Every user has a specific role in the system: Teachers and students. This role may be different for different courses in the same system. Develop a database with user data and provide a mechanism for user authentication.

**Known uses**

1. **SQLT-Web (Structured Query Language Tutor) [5]:** An intelligent tutoring system for teaching SQL (Structured Query Language). It acquires information about a student through a login screen. Individual student models are stored permanently on the server, and retrieved for each student's session.
2. **Cyberphysique [25]:** An intelligent tutoring system based on the World Wide Web that teaches physics. It acquires information about a student through a login screen. Individual student models are stored permanently on the server, and retrieved for each student's session.

**Related Patterns**

User Model Definition

**References**

The **Registration-Authentication-Access Control Pattern** is one of the patterns that were used in learning management systems [35], however, through our search for patterns in the field of ITS we discovered that this pattern can be useful to ITS developers and designers.

**4.3.2.2 Pattern 7: User Model Definition****Classification**

Adaptive Patterns

**Problem**

In a traditional educational setting, an instructor is considered a good one when he can take into account different learning styles and needs. When the instructor's role is to be played by an ITS this makes the system's adaptation to individual learning-related characteristics essential. What information should an ITS keep for the student in order to offer him the best possible learning

experience?

### **Motivation**

A student model is essentially the image the system has about the learner. An ITS enriches its functionality by maintaining a student model and providing mechanisms to modify application features based on that. Modifications can be related to the organizational and presentational issues of the learning resources. These modifications can be deciding to move to a higher difficulty level, showing the student more detailed feedback, etc., thus resulting in a personalized instruction. The closer the student model is to the learner's real characteristics and needs, the better the personalization. Therefore, the information kept in the student model has to be such that it describes the learner in the best way possible, but at the same time allows the model to be flexible in its manipulation.

Standardization of the student model is an important issue, because through this we can greatly enhance the student model's portability. This will allow learners to use several different ITS(s) and to carry their personal model with them, providing the systems with the same image of themselves, without that leading to compatibility problems. We need a student model that is small, compact and flexible.

### **Solution**

A student in general is very complex to describe, meaning that theoretically, the information that would be needed to fully describe him, would be too much for an application to handle, but also part of it would probably not be utilized. Consequently, a certain number of information items have to be carefully selected to form the student model. In an ITS setting, the items have to be directly related to the user as a learner – anything that would be considered useful to better adapt the learning experience to the learner's particular characteristics.

A complete user model definition should generally be comprised of the following elements:

1. **Demographic data**, which are relevant to the particular ITS (e.g. as age, gender, etc.)
2. **Student goals**, which are related to the long term and short term learning goals related to learning objectives of specific concepts to be learnt (e.g. "to complete course X")

3. **Student preferences**, with respect to the various dimensions of the learning opportunity (e.g. the mode of delivery, accessibility requirements, or assessment)
4. **Student knowledge**, which includes the knowledge level about concepts to be learned and weaknesses and strengths on particular areas, sections or points of the concepts.
5. **Usage data**, which include information like which pages were viewed, in what order, etc.
6. The **Stereotype** that applies to the student, which essentially is the group of learners he belongs to based on some predefined presuppositions in terms of knowledge level, learning and cognitive styles (e.g. the novice student, the intermediate student, the advanced student, the expert student stereotypes etc.).

#### Known Uses

1. **An Interactive Course Support for Greek [18]**: An intelligent tutoring system that teaches the Greek language to foreigners. It has three student stereotypes beginner, intermediate and advanced. Feedback messages are customized to suit current student knowledge level where beginners receive more detailed feedback than advanced students.
2. **The German Tutor [15]**: An intelligent tutoring system for teaching German. It has three student stereotypes beginner, intermediate and advanced. Feedback messages are customized to suit current student knowledge level where beginners receive more detailed feedback than advanced students.
3. **The Web Based German Tutor [21]**: A web based intelligent tutoring system for teaching German. It has three student stereotypes beginner, intermediate and advanced. Feedback messages are customized to suit current student knowledge level where beginners receive more detailed feedback than advanced students.
4. **Web Passive Voice Tutor (Web PVT) [17]**: An intelligent tutoring system for teaching English passive voice. It has four student stereotypes novice, beginner, intermediate and expert. Feedback messages are customized to suit current student knowledge level where beginners receive more detailed feedback than advanced students.

All the above mentioned ITSS records information about which concepts the student has mastered and to what extent.

**However**, all the above mentioned ITSs take into consideration the student stereotype , knowledge, stereotype, short term goals and usage data ,but none of them take into consideration student demographic data, long term goals or preferences.

However, in our implementation of the **Arabic Tutor**, which is described in detail in chapter 5, we took both short term and long term student goals into consideration when implementing both **PLITS** and The **Arabic Tutor**.

### **Related patterns**

Registration-Authentication-Access-Control

User Goals

New Student Model Initialization

### **References**

The **User Model Definition Pattern** is one of the patterns that were used in Adaptive systems [28], however, through our search for patterns in the field of ITS we discovered that this pattern can be useful to ITS developers and designers.

#### **4.3.2.3 Pattern 8: User Goals**

##### **Classification**

Adaptive Patterns

##### **Problem**

The student model description is a part of the student model component of an ITS and it should include student goals. What information should be considered as student goals in a student model that is to be used in an Intelligent Tutoring System?

##### **Motivation**

- Being able to model the student's educational goals can facilitate adaptation. An ITS can deliver the same course differently to learners with different educational goals, by setting the appropriate conditions to meet those goals. For example a learner with a goal "to master subject X" will receive more in-depth tutoring than a learner with a goal "to familiarize themselves with subject X".
- It is incorrect to assume that all users/learners aim at learning all of the material offered by an

ITS.

- Educational goals can vary in scope. For example they may refer to the whole duration of the course, or to only a part of it.
- There are some types of student goals that can be determined by the learners, but some others cannot. For example, the learners can determine the initial educational goal before starting a course using the ITS, but probably cannot be in a position to set the best (short-term) goals for themselves while the course is in progress.

### **Solution**

Include specific student goals in the student model in order to facilitate adaptation, and capture the real intent of the learner with respect to the learning material. The student goals can be divided into two categories:

**Long-term goals:** Educational goals that are valid for a longer period of time and require significant effort to be met. Long-term goals are usually determined by the learners.

**Short-term goals:** Educational goals that are valid for a shorter period of time and require relatively moderate effort to be met. Short term goals are usually determined by the **Tutor Module** component of an ITS.

In order to manage short term goals a goal-modeling component may be required that will take relevant data like student knowledge process it and derive goals.

### **Known uses**

**1. An Interactive Course Support for Greek [18]:** An intelligent tutoring system that teaches the Greek language to foreigners. It has three student stereotypes beginner, intermediate and advanced. Short term goals are accomplished by tailoring teaching material content and sequence to suit current student knowledge level.

**2. The German Tutor [15]:** An intelligent tutoring system for teaching German. It has three student stereotypes beginner, intermediate and advanced. Short term goals are accomplished by tailoring teaching material content and sequence to suit current student knowledge level.

**3. The Web Based German Tutor [21]:** A web based intelligent tutoring system for teaching German. It has three student stereotypes beginner, intermediate and advanced. Short term goals are accomplished by tailoring teaching material content and sequence to suit current student knowledge level.

**4. Web Passive Voice Tutor (Web PVT) [17].** It has four student stereotypes novice, beginner, intermediate and expert. Short term goals are accomplished by tailoring teaching material content and sequence to suit current student knowledge level.

**However,** all the above mentioned ITSs do not take into consideration the student's long term goals. Nevertheless, in our implementation of the **Arabic Tutor**, which is described in detail in chapter 5, we took both short term and long term student goals into consideration when implementing both **PLITS** and the **Arabic Tutor**.

#### **Related patterns**

Registration-Authentication-Access-Control

User Model Definition

New Student Model Initialization

Wizard

Shield

#### **References**

The **User Goals Pattern** is one of the patterns that were used in Adaptive systems [28], however, through our search for patterns in the field of ITS we discovered that this pattern can be useful to ITS developers and designers.

#### **4.3.2.4 Pattern 9: New Student Model Initialization**

##### **Classification**

Adaptive Patterns

##### **Problem**

The ITS initializes the student model before all interactions. What is the minimum amount of information needed, to start the system? What kind of information and what amount is the student capable or willing to provide?

## Motivation

Not all elements of the user model definition have to be acquired in order for the student to start interacting with the ITS. There are two reasons for that:

- In the beginning of an interaction session, students do not like to spend a lot of time providing information about them, answering long questionnaires for instance.
- It is not necessary to have a complete model of the student; a partial model (with proper selection of a subset of student model elements) will be acceptable.

There are student model elements that can be acquired directly from the student and elements that can be acquired automatically through the ITS. For instance, demographic data can only be provided by the student. On the other hand, user knowledge can be derived by the system e.g. via the prehistory of student's learning activities in other educational environments or through an entry exam.

It is also important to initialize the student stereotype, because the learning tasks are customized according to the stereotypes.

### There are two options to initialize the student model with knowledge:

- **It can be speculated.** This option is not recommended or encouraged by us although it is followed by some ITSs. The reason for that is there is no need to speculate when there is a way to accurately acquire the information that you need.
- **It can be identified with certainty.** This can be done through any of the methods that are described in the solution section. This option is encouraged by us.

## Solution

The ITS developers and designers need to initialize the student model with two types of knowledge:

- **Knowledge that is acquired from the students:** This knowledge should be

determined with the guidance of both the **User Model Definition Pattern** and the **User Goals Pattern**.

- **Knowledge that can be acquired automatically from the system:** This knowledge can be acquired by implementing an entry exam that uses an **Exam Generator Component**. This component can use the pool of questions that was filled earlier by the teachers through the **Early Bird Pattern** and the **Course Creation and Customization Pattern**, and randomly choose a set of questions that represent different difficulty levels thus can accurately measure current student knowledge level. After the student takes this entry exam, he will view his score sheet that indicates his current stereotype per topic. This stereotype is used to initialize the student model.

#### Known Uses

1. **Web PVT [17]:** The student is initially assigned to one of the four distinct stereotypes, namely novice, beginner, intermediate and expert, according to her/his performance on a preliminary test.
2. **Verb Expert [20]:** At the beginning of each session, the tutor starts the interaction with the student by presenting him by an exercise on a given topic. The Student Modeler compares the answer of the student with that of the expert Module in order to identify the errors and to formulate some hypotheses about their causes in order to initialize the student model.

#### Related patterns

User Model Definition

New Student Model Maintenance

User Goals

Whole Part

New Adapter

Master Slave

#### References

This pattern was based on the **User Model Initialization Pattern** that was used in Adaptive



systems [28], however, through our search for patterns in the field of ITS we discovered that this pattern can be modified to suit the needs of ITS developers and designers.

The **User Model Initialization Pattern** initializes the student model with the student stereotype through one of the following methods:

- User driven. For instance the user specifies explicitly that (s) he/she belongs to the novices' stereotype.
- Inferred by rules .For instance rules tell which UM elements and with what values can activate a stereotype.
- Speculated by rules .In this case there is absolutely no information which can suggest a certain stereotype, the developers should have some rules to allow selection of the stereotype. For instance, a rule of this kind might be: if user does not specify his/her knowledge level, then assume it is average.

We modified the solution suggested by the **User Model Initialization Pattern** in the **New Student Model Initialization Pattern** .The modification implemented by us depended on accurately initializing the student model by using an exam generator component that automatically generates an entry exam after student sign up. Through this entry exam the student stereotype will be determined automatically and accurately instead of speculation or inferring as suggested in the **User Model Initialization Pattern**.

#### 4.3.2.5 Pattern 10: New Student Model Maintenance

##### Classification

Adaptive Patterns

##### Problem

During the course of interaction, many things about the student are changed, e.g. current student knowledge level. Thus, the student model must be adapted to the new realities. How should the system capture those changes so as to maintain a good student model [28]?

## Motivation

The assumption that the student model will remain the same as when it was acquired originally is in most cases incorrect. As in tutoring between a human tutor and a student, where the student constantly demonstrates changes, the student of an ITS also changes and as a result his model has to reflect this. During the course of interaction, leverage of student knowledge develops and builds up. Since the adaptation is to a large extent based on student knowledge and usage data, changes should definitely be recorded and be related to a "cause / result" [28].

In fact, information such as demographic data does not change with a high frequency. However, information like topics covered and student knowledge level per topic changes continuously [28].

It is also important for users to be in control, to a degree acceptable to the ITS, of their model for several reasons. They need to be able to modify information in their model if they feel that it is inaccurate or incorrect. Also, being in control, builds up their trust in the system [28].

## Solution

Methods used for maintenance of an accurate student model:

- **User Driven:** In this method the student provides explicit information about changes in his student model.
- **System Driven:** In this method the ITS derives information by closely monitoring the performance of the student.

The ITS designers should define the conditions that govern the maintenance of the student model. In particular the designer should define the scope of the maintenance changes. The scope defines the reason for updates. The reason is then quantified in terms of choice of elements to undergo change.

### Methods used to update the student model description:

- The student model maintenance module can use an exam generator component to conduct a per topic exam. This can occur after the student is presented by the topic

learning material. The exam generator component can use the pool of questions supplied by the subject instructor to randomly select a number of questions that can test the current student knowledge level and according to the results of this exam the user model maintenance module can update the student model to reflect his current status and the topics covered and his knowledge level per topic.

- The student model maintenance module can take advantage of the agent techniques by using an agent that monitors the student performance per topic and reaches a set of conclusions, for example, that the student performance improves more if he is presented by a set of examples, or that the student performance is excellent and he doesn't need any more examples and may proceed to the final exam.

#### Known uses

1. **Web PVT [17]:** A web based intelligent tutoring system for teaching the English passive voice. It records information about which concepts the student has mastered and to what extent. In addition, it records the kinds of error the student has made during past interactions as well as the most suitable explanation of each category of error. The information from the long term student model forms an individual model of the student, which together with the active stereotype are used in order to provide adaptive navigation support and perform intelligent analysis of the student's solutions to exercises.
2. **An Interactive Course Support for Greek [18]:** A web based intelligent tutoring system for teaching Greek. The student model is a representation of the current skill level of the student. For each student the student model keeps score across a number of error types, or nodes, for example, grammar or vocabulary. The score for each node increases and decreases depending on the grammar's analysis of the student's performance. The amount by which the score of each node is adjusted is specified in a master file and may be weighted to reflect different pedagogical purposes.

#### Related patterns

New Student Model Initialization

New Adapter

Master Slave

## References

This pattern was based on the **User Model Maintenance Pattern** that was used in Adaptive systems [28], however, through our search for patterns in the field of ITS we discovered that this pattern can be modified to suit the needs of ITS developers and designers.

Methods used to track changes in the user model in the **User Model Maintenance Pattern**:

- Using a form that the user fills that acts as a questionnaire to indicate the amount of benefit he/she gained from using the system.
- Interactive update of the user model. For instance by showing a pop-up form requesting the user to explicitly answer a question.

We modified these methods in the **New Student Model Maintenance Pattern** and included a number of more efficient methods that were included in the solution section and that we will list again briefly:

- The student model maintenance module can use an exam generator component to conduct a per topic exam this exam will be automatically generated and graded and it will record the changes in student stereotype in his student model.
- The student model maintenance module can take advantage of the agent techniques by using an agent that monitors the student performance per topic and reaches a set of conclusions.

### 4.3.2.6 Pattern 11: New Adapter Also Known as Wrapper

#### Classification

Structural Patterns

#### Intent

Convert the interface of a class into another interface that clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces [7].

## Motivation

Sometimes a toolkit class that's designed for reuse isn't reusable only because its interface doesn't match the domain specific interface an application requires.

The **New Adapter Pattern** is capable of converting the interface of a class into another interface that the client expects. Often the Adapter is responsible for functionality the adapted class doesn't provide [7].

ITS developers and designers prepare teaching material that should be studied by students.

**In any ITS there is an aggregation relationship that is illustrated as follows:**

- Teaching Material is composed of a number of topics.
- Each Topic is composed of a number of learn items.
- Each Topic has a number of exercises related to it.
- Each Learn Item has a number of examples related to it.

**There are two problems that the New Adapter Pattern can help us in solving:**

- **How can we represent each student's specific status per topic?**
- **How can we represent each student's specific status per exercise?**

**How can we represent each student's specific status per topic?**

There is a problem with the Topic class. This problem occurs because this class doesn't match the domain specific interface that the application requires because it doesn't reflect the student specific status per topic.

**This class doesn't represent the following information:**

- The student stereotype per topic; whether he is a Beginner, Intermediate, Advanced or Expert student.
- What learning item is currently being learned inside the topic by the student?
- How many times the student viewed a certain topic?
- What are the exams that he took on a certain topic?
- Student grades on every exam concerning this topic.

That is where the **New Adapter Pattern** fits in ITSs Design. An Adapter class is needed that is capable of converting the interface of a class (Topic) into another interface that the client

expects (StudentTopic). And to provide us with the functionality that the adapted class (Topic) doesn't provide.

### **How can we represent each student's specific status per exercise?**

There is a problem with the Exercise class. This problem occurs because this class doesn't match the domain specific interface that the application requires because it doesn't reflect the student specific status per exercise.

### **This class doesn't represent the following information:**

- Did the student view this exercise or not?
- The student answer on this exercise and whether he answered correctly or not, this will help to understand any misconception that the student might have.
- The student score per exercise.
- The number of student tries for solving this exercise.
- Did the student use out all his tries in this exercise or not yet?
- The examples that is related to this exercise, this means that if the student answers incorrectly he is shown some teaching material to revise the learning items that is covered by this exercise, this teaching material includes some examples so we should keep a record of the examples that is related to this exercise.

That is where the **New Adapter Pattern** fits in ITSS. A StudentExercise class is needed to act as an Adapter that is capable of converting the interface of the Exercise class into another interface that the client expects. And to provide us with the functionality that the adapted class (Exercise) doesn't provide:

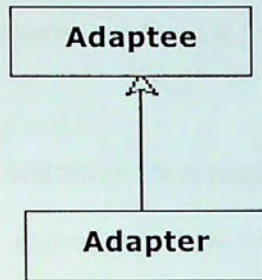
### **Applicability**

#### **Use the New Adapter Pattern when [7]:**

- You want to use an existing class, and its interface does not match the one you need.
- You want to create a reusable class that cooperates with unrelated or unforeseen classes, that is, classes that don't necessarily have compatible interfaces.

Structure

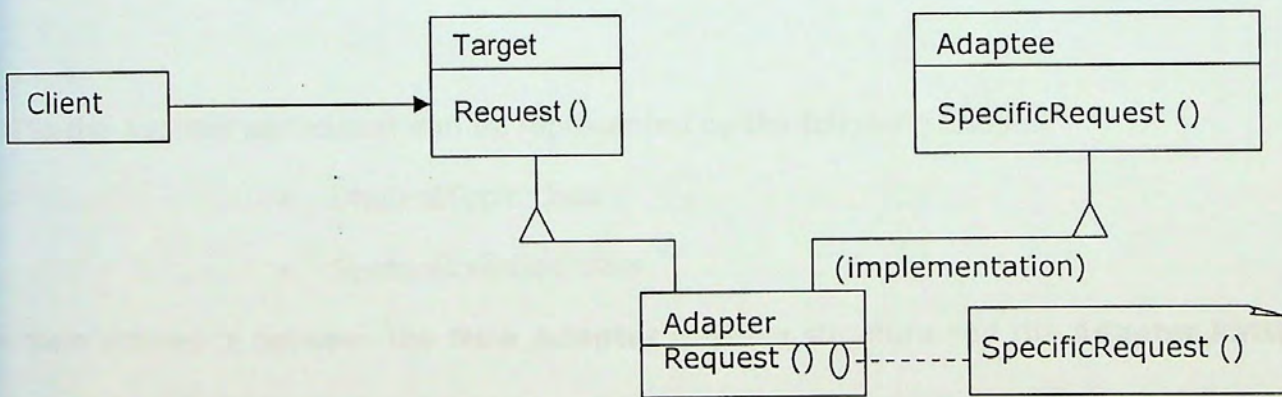
The **New Adapter Pattern** Structure is illustrated in figure (4-7).



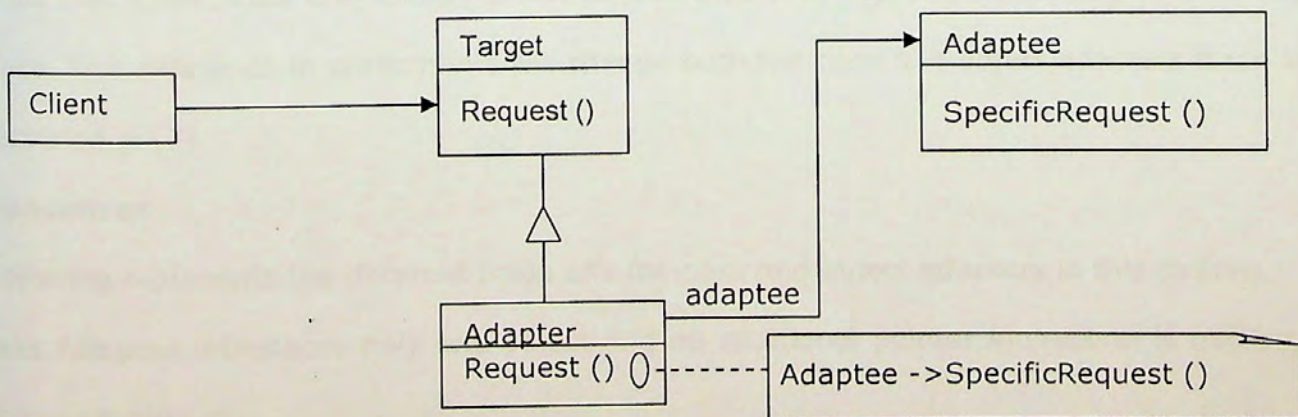
**Figure (4-7): New Adapter Pattern Structure**

The reason for naming this pattern New Adapter was that it is a variant of the **Adapter Pattern** whose structure is represented in figure (4-8).

A class adapter uses multiple inheritances to adapt one interface to another:



An object adapter relies on object composition:



**Figure (4-8): Adapter Pattern Structure [7]**

## Participants

In the context of ITS we have the following participants:

- **Client**
  - Collaborates with objects conforming to the Target interface.
- **Adaptee**
  - Defines an existing interface that needs adapting.

In ITSs the Adaptee participant can be represented by the following classes:

- Topic class.
- Exercise class.

- **Adapter**
  - Adapts the interface of Adaptee to the Target interface. It also acts as the Target.

In ITSs the Adapter participant can be represented by the following classes:

- StudentTopic class
- StudentExercise class

The main difference between the **New Adapter Pattern** structure and the **Adapter Pattern** is that the **Adapter Pattern** has two extra participants Target; which defines the domain specific interface that Client uses and Client; which collaborates with objects conforming to the Target interface. This difference in participants will change both the class and object adapters trade offs as illustrated in [7].

## Consequences

The following represents the different trade offs for class and object adapters in this pattern.

**A Class Adapter** introduces only one object and no additional pointer indirection is needed to get to the adaptee.

**An Object Adapter** makes it harder to override Adaptee behavior. It will require sub classing Adaptee and making Adapter refer to the subclass rather than the Adaptee itself.

## Implementation



The implementation of the **New Adapter Pattern** is usually straight forward. The following steps should be followed:

1. **Implement the Adapter Class.** Which in the field of ITS can be represented by :
  - Topic Class.
  - Exercise Class.
2. **Implement the Adaptee Class.** Which in the field of ITS can be represented by:
  - StudentTopic Class.
  - Student Exercise Class.

### Sample Code

Please refer to appendix D for sample source code.

### Related Patterns

New Student Model Initialization

Whole Part

Master Slave

New Strategy

New Student Model Maintenance

Builder

### References

This pattern was based on the **Adapter Pattern** [7]. However, through our search for patterns in the field of ITS we discovered that this pattern can be modified to suit the needs of ITS developers and designers.

#### 4.3.3 Phase Three: Building the Tutor Module

The following patterns are needed in building the tutor module.

##### 4.3.3.1 Whole Part Pattern

The **Whole Part Pattern** was mentioned earlier in section 4.3.1.2. **The Whole Part Pattern**

**can be used in the Tutor Module in two situations:**

- Lesson Presentation Planner.
- Exam Generator Component.

This will be illustrated in more details in chapter 5.

#### 4.3.3.2 Pattern 12: New Strategy Also Known as Policy

##### Classification

Behavioral Pattern

##### Intent

Define a family of algorithms, encapsulate each one, and make them interchangeable. **New Strategy Pattern** lets the algorithm vary independently from clients that use it [7].

##### Motivation

**New Strategy Pattern** is needed when an algorithm uses data that clients shouldn't know about. This pattern is used to avoid exposing complex, algorithm specific data structures [7].

In the context of ITS the sequence number indicates the successor and predecessor relationship between any two objects. The **New Strategy Pattern** can be used in defining a sorting algorithm; it can be used to sort multiple objects:

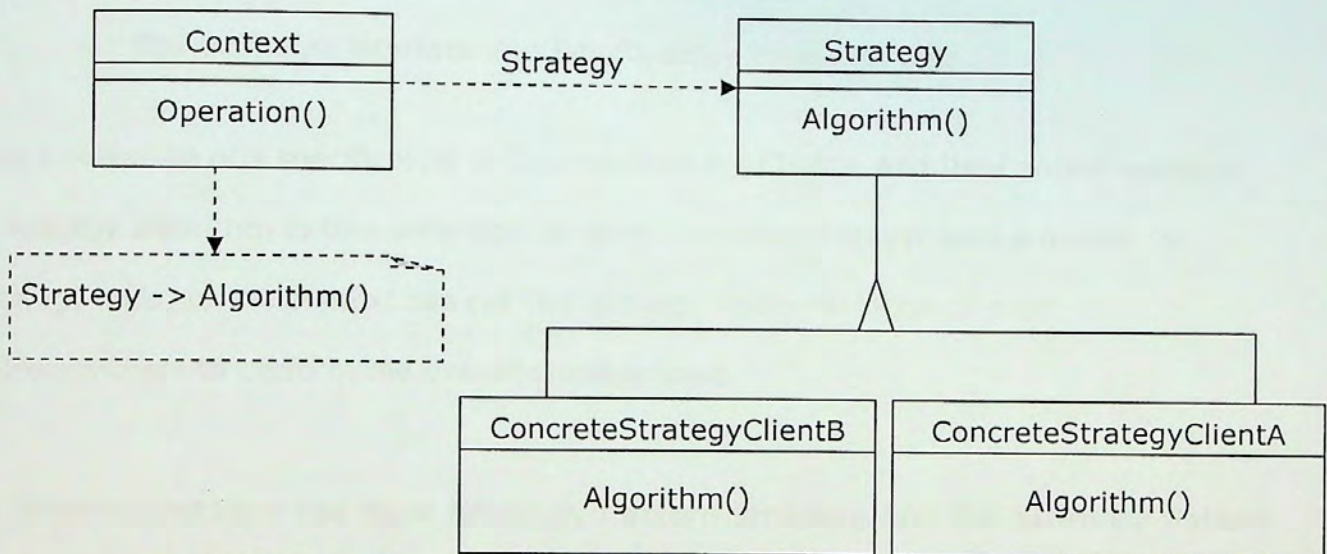
- **Topics, learn items, exercises, examples:** These objects are sorted according to sequence number. The sequence number is entered initially by the teacher using both the **Early Bird Pattern** and the **Course Creation and Customization Pattern**.
- **Exams:** At the end of the learning session the student is presented with a score sheet that shows his score in all the exams that he took, in that report exams should be sorted. Exams are usually sorted by date.
- **Student's Topics:** Sorted according to both the sequence and the student's stereotype per topic. This means that after we give the student the entry exam, according to his score in the exam we determine the list of topics that needs to be learned by him. If he needs to learn two topics that has the same sequence number then we will start with the topic that he has the lowest stereotype in (if both topics has the same sequence number yet the student level is beginner in one topic and intermediate in the other topic we usually start with the topic in which the student is a beginner).

**Applicability**

**Use New Strategy Pattern when [7]:**

- Use the **Strategy Pattern** to avoid exposing complex, algorithm specific data structures that the client should not worry about.
- A class defines many behaviors, and these appear as multiple conditional statements in its operations. Instead of many conditionals, move related conditional branches into their own Strategy class.

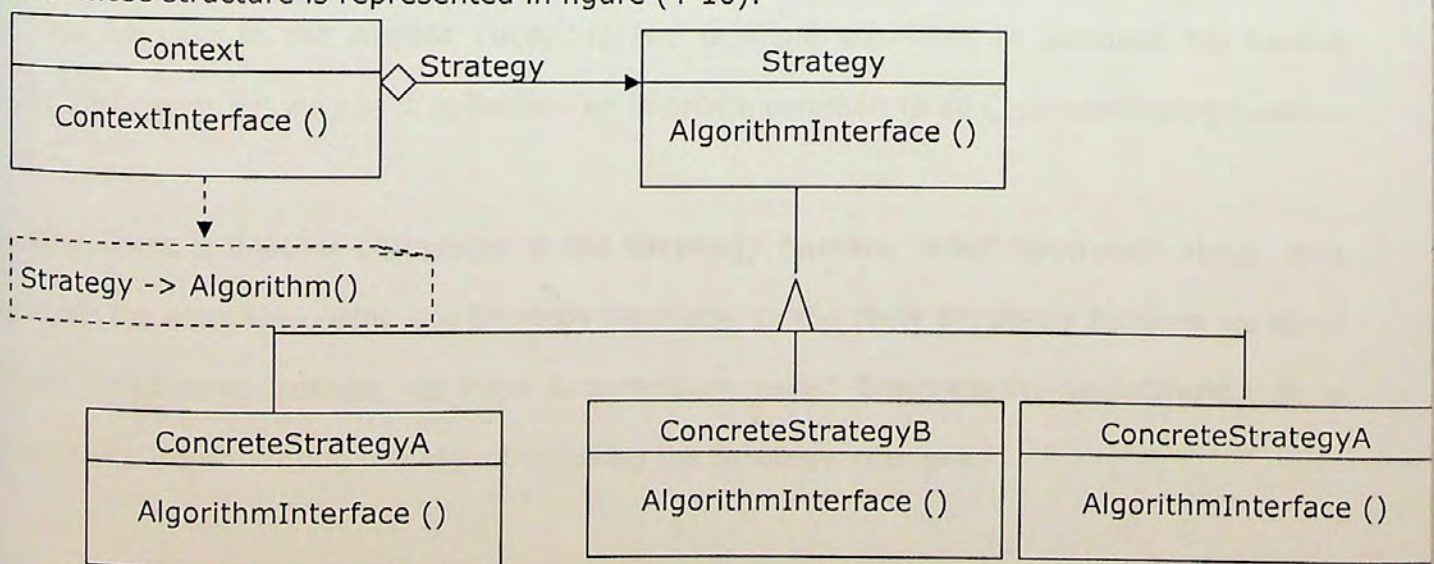
**Structure**



**Figure (4-9): New Strategy Pattern Structure**

The reason for naming this pattern New Strategy was that it is a variant of the **Strategy**

**Pattern** whose structure is represented in figure (4-10):



**Figure (4-10): Strategy Pattern Structure [7]**

## Participants

### Strategy

The Strategy Participant declares an interface common to all ConcreteStrategyClients.

### ConcreteStrategyClient

The ConcreteStrategyClient Participant implements the algorithm using the Strategy interface.

### Context

- Is configured with a ConcreteStrategy object.
- Maintains a reference to a Strategy object.
- May define an interface that lets Strategy access its data.

Context has a collection of a specific kind of ConcreteStrategyClients, and the Context wants to apply the Strategy algorithm to this collection, so each ConcreteStrategyClient provides its specific strategy logic, so the context can call the strategy implementation of each ConcreteStrategyClient to perform the overall strategy logic.

The main difference between the **New Strategy Pattern** structure and the **Strategy Pattern** can be illustrated as follows:

The Strategy Participant in the **Adapter Pattern** declares an interface common to all supported algorithms however in the **Arabic Tutor** we had a single algorithm so although we have a Strategy Participant yet we use it to declare an interface common to all ConcreteStrategyClients.

In addition there is another participant in the **Strategy Pattern** called ConcreteStrategy; that implements the algorithm using the Strategy interface. In the **New Strategy Pattern** we don't have this participant. Instead, we have a participant called **ConcreteStrategyClient** that is responsible for implementing the algorithm using the Strategy interface.

This difference in participants will change implementation and consequences as illustrated in [7].

## Consequences

### The New Strategy Pattern has the following benefits and drawbacks:

1. The **New Strategy Pattern** offers an alternative to sub classing. Inheritance offers another way to support a variety of algorithms or behaviors. You can subclass a Context class directly to give it different behaviors. But this hard wires the behavior into Context harder to understand, maintain, and extend. And you can't vary the algorithm dynamically. You wind up with many related classes whose only difference is the algorithm or behavior they employ. Encapsulating the algorithm in separate Strategy classes lets you vary the algorithm independently of the context, making it easier to switch, understand, and extend.
2. Strategies eliminate conditional statements. The **New Strategy Pattern** offers an alternative to conditional statements for selecting desired behavior. When different behaviors are lumped into one class, it's hard to avoid using conditional statements to select the right behavior. Encapsulating the behavior in separate Strategy classes eliminate these conditional statements. Code containing many conditional statements often indicates the need to apply **New Strategy Pattern**.

## Implementation

### Consider the following implementation issues:

1. Defining the Strategy and Context Interfaces. The Strategy and Context interfaces must give a ConcreteStrategy efficient access to any data it needs from a context, and vice versa.
  - One approach is to have Context pass data in parameters to Strategy operations. This keeps Strategy and Context decoupled. On the other hand, Context might pass data the Strategy doesn't need.
  - Another technique has a context pass itself as an argument, and the strategy requests data from the context explicitly. Alternatively, the Strategy can store a reference to its context, eliminating the need to pass anything at all.
2. Making Strategy objects optional. The Context class may be simplified if it's meaningful

not to have a Strategy object. Context checks to see if it has a Strategy object before accessing it. If there is one, then Context uses it normally. If there isn't a strategy, then Context carries out default behavior. The benefit of this approach is that clients don't have to deal with Strategy objects at all unless they don't like the default behavior.

### Sample Code

Please refer to appendix D for sample source code.

### Related Patterns

Whole Part

New Adapter

New Student Maintenance

Course Creation and Customization

Observer

### References

This pattern was based on the **Strategy Pattern** [7], however, through our search for patterns in the field of ITS we discovered that this pattern can be modified to suit the needs of ITS developers and designers.

#### 4.3.3.3 Pattern 13: Template Method

##### Classification

Behavioral Patterns

##### Intent

Define the skeleton of an algorithm in an operation, deferring some steps to subclasses.

Template Method subclasses redefine certain steps of an algorithm without changing the algorithm's structure [7].

##### Motivation

A Template Method defines an algorithm in terms of abstract operations that subclasses override to provide concrete behavior [7].

Any ITS contains a number of questions or exercises that aim to test the current knowledge level of the student per topic. These questions differ in their type.

**The most common types of exercises that are used in the ILTS domain are:**

- Multiple Choice Questions(MCQ)
- Fill-in-the-Blank
- True/False
- Dictation
- Build a Phrase
- Build a Sentence
- Guess the Word
- Find the Word
- Which Word is Different
- Word Order Practice
- Check-and correct
- Rewrite in another voice(active or passive)

We need to implement a method that automatically shows the title of the question at run time according to object type.

### **Applicability**

**The following situations require the usage of the Template Method Pattern [7]:**

- To implement the invariant parts of an algorithm once and leave it up to subclasses to implement the behavior that can vary.
- When common behavior among subclasses should be factored and localized in a common class to avoid code duplication. First, the differences in the existing code should be identified and then the differences should be separated into new operations. Finally, the differing code should be replaced with a Template Method that calls one of these new operations.

Structure

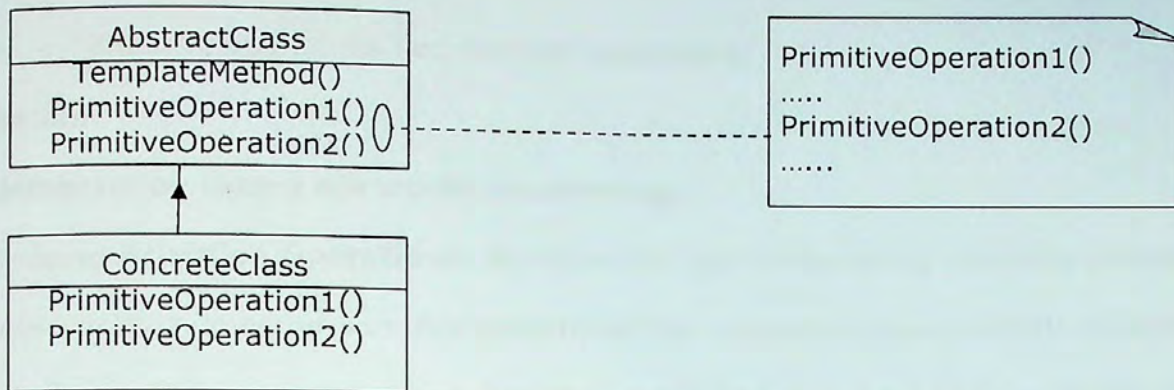


Figure (4-11): Template Method Pattern Structure [7]

Participants

- **Abstract Class**

- Defines abstract **primitive operations** that concrete subclasses define to implement steps of an algorithm [7].
- Implements a Template Method defining the skeleton of an algorithm. The Template Method calls primitive operations as well as operations defined in Abstract Class or those of other objects [7].

- **Concrete Class**

Implements the primitive operations to carry out subclass specific steps of the algorithm [7].

**Consequences**

Template Methods are fundamental techniques for code reuse .They are particularly important in class libraries, because they are the means for factoring out common behavior in library classes [7].

Template Methods lead to an inverted control structure that's sometimes referred to as "Don't call us, we'll call you". This refers to how a parent class calls the operations of a subclass and not the other way around [7].

**Template Methods call the following three kinds of operations [7]:**

- Concrete operations (either on the Concrete Class or on client classes).
- Concrete Abstract Class operations (i.e., operations that are generally useful to



subclasses).

- o Primitive operations(i.e., abstract operations)

## Implementation

These implementation issues are worth considering:

1. **Minimizing Primitive Operations:** An important goal in designing template methods is to minimize the number of primitive operations that a subclass must override to flesh out the algorithm. The more operations that need overriding, the more tedious things get for clients.
2. **Naming Conventions:** The operations that should be overridden can be identified by adding a prefix to their names. For example "Do".

## Sample Code

Please refer to appendix D for sample source code.

## Related Patterns

Whole Part

## References

The **Template Method Pattern** is one of the Gang of Four Patterns [7].

### 4.3.3.4 Pattern 14: Master Slave

#### Classification

Design Patterns

#### Intent

The **Master Slave Pattern** supports fault tolerance and computational accuracy. A master component distributes work to identical slave components and computes a final result from the results these slaves return.

#### Motivation

Divide and Conquer is a common principle for solving many kinds of problems. The result of the whole calculation is computed from the results provided by each partial process. Several forces arise when implementing such a structure:

- Clients should not be aware that the calculation is based on the divide and conquer principle.
- Neither clients nor the processing of sub tasks should depend on the algorithms for partitioning work and assembling the final result.

### Applicability

This **Master Slave Pattern** can be used in the following application areas:

- **Fault Tolerance.** The execution of a service is delegated to several replicated implementations. Failure of service executions can be detected and handled.
- **Computational Accuracy.** The execution of a service is delegated to several different implementations. Inaccurate results can be detected and handled.

Provide all slaves with a common interface. Let clients of the overall service communicate only with the master.

In the **ITS domain Master Slave Pattern** can be used to compute the exam results of each student according to their grade in each particular exercise within the exam.

### Structure

The structure of the **Master Slave** can be explained as follows:

1. The master component provides a service that can be solved by applying divide and conquer principle. It offers an interface that allows clients to access this service.
2. The master implements functions for partitioning work into several equal sub tasks, starting and controlling their processing, and computing a final result from all the results obtained.
3. The master maintains references to all slave instances to which it delegates the processing of sub tasks.
4. The slave component provides a sub service that can process the sub tasks defined by the master. Within a Master Slave structure, there are at least two instances of the slave component connected to the master.

## Participants

The Master Slave Participants is illustrated from the following tables:

**Table (7): Master Class Responsibility and Collaborators [12]**

Class	Collaborators
Master	<ul style="list-style-type: none"> <li>Slave</li> </ul>
Responsibility	
<ul style="list-style-type: none"> <li>Partitions work among several slave components</li> <li>Starts the execution of slaves</li> <li>Computes a result from the sub result the slaves return.</li> </ul>	

**Table (8): Slave Class Responsibility and Collaborators [12]**

Class	Collaborators
Slave	
Responsibility	
<ul style="list-style-type: none"> <li>Implements the sub service used by the master.</li> </ul>	

## Consequences

The Master Slave Pattern provides several benefits:

1. **Exchangeability and extensibility.** By providing an abstract slave class, it is possible to exchange existing slave implementations or add new ones without major changes to the master. Clients are not affected by such changes.
2. **Separation of concerns.** The introduction of the master separates slave and client code from the code for partitioning work, delegating work to slaves, collecting the results from the slaves, computing the final result and handling slave failure or inaccurate slave results.

## Implementation

The implementation of the Master Slave Pattern follows five steps:

1. **Divide Work.** Specify how the computation of the task can be split into a set of equal sub tasks. Identify the sub services that are necessary to process a subtask.
2. **Combine sub task results.** Specify how the final result of the whole service can be computed with the help of the results obtained from processing individual sub tasks.

3. **Implement the slave components.**

4. **Implement the master.**

The master computes a final result with help of the results collected from the slaves.

#### **Sample code**

Please refer to appendix D for sample source code.

#### **Related Patterns**

New Adapter

Observer

New Student Model Maintenance

New Student Model Initialization

#### **References**

The **Master Slave Pattern** is one of the POSA patterns [12].

#### **4.3.3.5 Singleton Pattern**

The **Singleton Pattern** was mentioned earlier in section 4.3.1.3. **ITS designers and developers can benefit from the Singleton Pattern in the Tutor Module** in the instantiation of collective student information as we will illustrate later in chapter 5.

#### **4.3.3.6 Pattern 15: Observer Also Known as Dependents, Publish Subscribe**

##### **Classification**

Behavioral Patterns

##### **Intent**

Define a one to many dependencies between objects so that when one object changes state, all its dependents are notified and updated automatically [7].

##### **Motivation**

A common side effect of partitioning a system into a collection of cooperating classes is the need to maintain consistency between related objects. However, we don't want to achieve consistency by making the classes tightly coupled, because that reduces their reusability [7].

The key objects in the **Observer Pattern** are Subject and observer. A subject may have a

number of dependent observers. All observers are notified whenever the subject undergoes a change in state. In response, each observer will query the subject to synchronize its state with the subject's state [7].

This kind of interaction is also known as publish subscribe. The subject is the publisher of notifications. It sends out these notifications without having to know who its observers are. Any number of observers can subscribe to receive notifications [7].

**In the ITS domain** an Observer class is needed to observe the status of all students and update the reports with the new student status as he navigates through the learning path. Thus if the teacher is monitoring the status of all students he will always see the up-to-date results of all students even if a student has just taken an exam.

### **Applicability**

Use the **Observer Pattern** in any of the following situations[7]:

- When an abstraction has two aspects, one dependent on the other. Encapsulating these aspects in separate objects lets you vary and reuse them independently.
- When a change to one object requires changing others, and you don't know how many objects need to be changed.
- When an object should be able to notify other objects without making assumptions about who those objects are. In other words, you don't want these object tightly coupled.

### **Structure**

The **Observer Pattern** Structure is illustrated in figure (4-12).

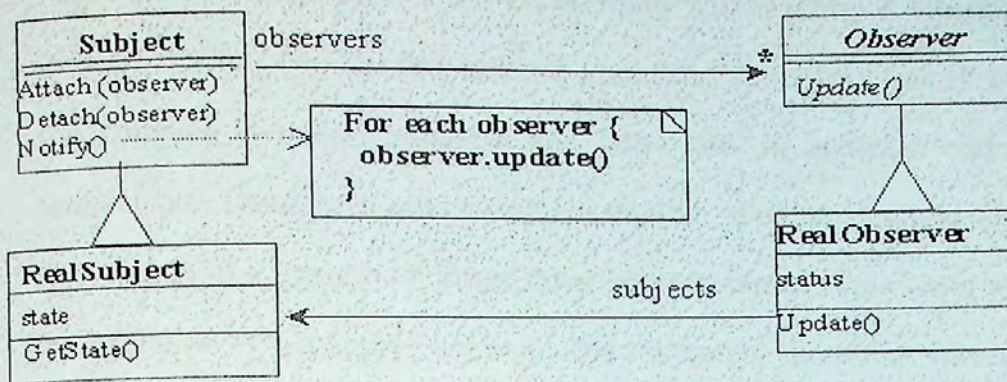


Figure (4-12): Observer Pattern Structure [7]

Participants [7]

- **Subject**
  - Knows its Observers. Any number of Observer objects may observe a subject.
  - Provides an interface for attaching and detaching Observer objects.
- **Observer**
  - Defines an updating interface for objects that should be notified of changes in a subject.
- **Concrete Subject**
  - Store state of interest to Concrete Observer objects.
  - Sends a notification to its Observers when its state changes.
- **Concrete Observer**
  - Maintains a reference to Concrete Subject object.
  - Stores state that should stay consistent with the subject's.
  - Implement the Observer updating interface to keep its state consistent with the subject's.

Consequences

The **Observer Pattern** lets you vary subjects and Observers independently. Subjects can be reused without reusing their Observers, and vice versa. Observers can be added without modifying the subject or other Observers [7].

Further benefits and liabilities of the Observer Pattern include the following [7]:

1. **Abstract coupling between Subject and Observer.** All a subject knows is that it

has a list of Observers, each conforming to the simple interface of the abstract Observer class. The subject doesn't know the concrete class of any Observer. Thus the coupling between subjects and Observers is abstract and minimal. Because Subject and Observers aren't tightly coupled, they can belong to different layers of abstraction in a system. A lower level subject can communicate and inform a higher level Observer, thereby keeping the system's layering intact. If Subject and Observer are lumped together, then the resulting object must both span two layers (and violate the layering), or it must be forced to live in one layer or the other (which might compromise the layering abstraction).

2. **Support for broadcast communication.** Unlike an ordinary request, the notification that a subject sends needn't specify its receiver. The notification is broadcast automatically to all interested objects that subscribed to it. The Subject doesn't care how many interested objects exist; its only responsibility is to notify its Observers. This gives you the freedom to add and remove Observers at any time. It's up to the Observer to handle or ignore a notification.
3. **Unexpected updates.** Because Observers have no knowledge of each other's presence they can be blind to the ultimate cost of changing the subject. A seemingly innocuous operation on the subject may cause a cascade of updates to Observers and their dependent objects. Moreover, dependency criteria that aren't well defined or maintained usually lead to spurious updates, which can be hard to track down.

This problem is aggravated by the fact that the simple update protocol provides no details on what changed in the subject. Without additional protocol to help Observers discover what changed, they may be forced to work hard and to deduce the changes.

### Implementation

Several issues related to the implementation of the dependency mechanism are discussed in this section [7]:

1. **Mapping subjects to their observers.** The simplest way for a subject to keep track of the observers it should notify is to store references to them explicitly in the subject.

However, such storage may be too expensive when there are many subjects and few observers. One solution is to trade space for time by using an associative look up to maintain the subject to observer mapping. Thus a subject with no observers does not incur storage overhead. On the other hand, this approach increases the cost of accessing the observers.

2. **Observing more than one subject.** It might make sense in some situations for an observer to depend on more than one subject. It's necessary to extend the Update interface in such cases to let the observer know which subject is sending the notification. The subject can simply pass itself as a parameter in the Update operations, thereby letting the observer know which subject to examine.
3. **Who triggers the update?** The subject and its observers rely on the notification mechanism to stay consistent. But what objects actually calls Notify to trigger the update? **Here are two options:**
  - a. Have state setting operations on Subject call Notify after they change the subject's state. The advantage of this approach is that clients don't have to remember to call Notify on the subject. The disadvantage is that several consecutive operations will cause several consecutive updates, which may be inefficient.
  - b. Make clients responsible for calling Notify at the right time. The advantage here is that the client can wait to trigger the update until after a series of state changes has been made, thereby avoiding needless intermediate updates. The disadvantage is that clients have an added responsibility to trigger the update. That makes errors more likely, since clients might forget to call Notify.
4. **Making sure Subject state is self consistent before notification.** It's important to make sure Subject state is self consistent before calling Notify, because observers query the subject for its current state in the course of updating their own state. This self consistency rule is easy to violate unintentionally when Subject subclass operations call inherited operations.



5. **Specifying modifications of interest explicitly.** Update efficiency can be improved by extending the subject's registration interface to allow registering observers only for specific events of interest. When such an event occurs, the subject informs only those observers that have registered interest in that event. One way to support this uses the notion of aspects for Subject objects.

### Sample Code

Please refer to appendix D for sample source code.

### Related Patterns

Master Slave

New Strategy

Singleton

### Anti-patterns

1. Beginners tend to connect an Observer directly to an observable so that they both have references to each other. However, this means that we cannot plug in a different Observer [7].
2. Another mistake is to make the Observers subclasses of the observable. This will not work because then each Observer is at the same time an observable. It is not therefore possible to have more than one Observer for an observable [7].

### References

The **Observer Pattern** is one of the Gang of Four Patterns [7].

#### 4.3.4 Phase Four: Building the Graphical User Interface Module

The following patterns are needed in building the graphical user interface module.

##### 4.3.4.1 Singleton Pattern

The **Singleton Pattern** was mentioned earlier in section 4.3.1.3. **ITS designers and developers can benefit from the Singleton Pattern in the GUI module in** the instantiation of the user interface as we will illustrate later in chapter 5.

#### 4.3.4.2 Pattern 16: Course Creation and Customization

##### Classification

Instructional Patterns

##### Problem

How can the instructors be assisted in building on-line courses in ITS so that some of the tasks they need to perform can be automated?

##### Motivation

ITS need to make the job of instructors easier by providing them with easy-to-use tools for creating, and customizing their courses so that they won't have to spend too much time and effort in performing those tasks. This way, courses will not be created from scratch, but instead instructors will reuse some design templates and easily perform generic activities and let the ITS take care of the details.

##### Solution

Provide the instructors with appropriate tools for creating a course and customizing it according to their preferences. The creation of courses should be based on design templates with pre-set interfaces, content structure and features or based on existing courses.

##### Known Uses

1. **Cyberphysique [25]:** An intelligent tutoring system based on the World Wide Web that teaches physics. This ITS provides an authoring environment dedicated to teachers and pedagogical designers. This authoring environment is used to computerize curriculum elements. Cyberphysique aims to decrease teachers' and pedagogical designers' workload by helping them to easily create and customize the course teaching material that acts as the nucleus of the domain module.
2. **Eon Tools [33]:** "Eon" is the name for a suite of authoring tools for building intelligent tutoring systems. Eon includes tools for authoring all aspects of intelligent tutors, including the learning environment, the domain knowledge, the teaching strategies, and the student model. Curriculum can be easily extended or modified to update information and theories, and new curriculum can be uploaded over the World Wide Web. Teachers

using the tutorials in their classroom can easily customize certain aspects of the tutorial to fit their needs (for example, by altering prerequisite relationships among topics or replacing pictures with ones more relevant to their classroom environment). In

### Related Patterns

Early Bird

Whole Part

New Strategy

Warning

### References

The **Course Creation and Customization Pattern** is one of the patterns that were used in learning management systems [35], however, through our search for patterns in the field of ITS we discovered that this pattern can be modified to suit the needs of ITS developers and designers.

The following patterns focus on solutions to problems that end users have when interacting with systems. These patterns take an end-user perspective which leads to a format where usability is the essential design quality. When taking the user perspective it becomes important to put emphasis on the argumentation for *how* and *why* the usability is improved. Without such a rationale it is impossible to see whether or why the solution is actually a *good* and *accepted* solution. Usability must be measurable in usage indicators [30].

**Usability can be measured with the following usage indicators [30]:**

- Learn-ability.
- Memor-ability.
- Speed of performance.
- Error rate.
- Satisfaction.
- Task completion.

Each pattern should therefore state the impact on these usage indicators. In short, if a pattern does not improve at least one usage indicator, it is *not* a user interface design pattern [30].

### **Categorizing User Problems**

Our patterns are task related and in this collection we categorize them according to the kind of usage problems they address.

### **Usability Principles [30]:**

1. **Visibility (User Guidance).** Gives the user the ability to figure out how to use something just by looking at it.
2. **Affordance.** Involves the perceived and actual properties of an object that suggest how the object is to be used.
3. **Natural mapping.** Creates a clear relationship between what the user wants to do and the mechanism for doing it.
4. **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.
5. **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.
6. **Feedback.** Indicates to the user that a task is being done and that the task is being done correctly.

These principles assume that users always exhibit fully rational behavior. In practice, users make mistakes and do things they do not really want to do.

### **Therefore, additional principles are [30]:**

7. **Safety.** The user needs to be protected against unintended actions or mistakes.
8. **Flexibility.** Users may change their mind and each user may do thing differently.

#### 4.3.4.3 Pattern 17: Wizard

##### Classification

User Interface Patterns (Interaction Patterns).

##### Problem

The student wants to perform an infrequent complex task consisting of several subtasks where decisions need to be made in each subtask which may not be known to the user. However, the number of subtasks must be small e.g. typically between  $\sim 3$  and  $\sim 10$  [30].

##### Usability Principle

User Guidance (Visibility) [30].

##### Motivation

- The user wants to reach the overall goal but may not be familiar or interested in the steps that need to be performed [30].
- The subtask can be ordered but are not always independent of each other i.e. a certain task may need to be finished before the next task can be done [30].
- To reach the goal, several steps need to be taken but the exact steps required may vary because of decisions made in previous steps [30].

##### Solution

Take the user through the entire task one step at the time. Let the user step through the tasks and show him which steps exist and which have been completed [30].

When the complex task is started, the user is informed about the goal that will be achieved and the fact that several decisions are needed. The user can go to the next task by using a navigation widget (for example a button). If the user cannot start the next task before completing the current one, feedback is provided indicating the user cannot proceed before completion (for example by disabling a navigation widget). The user is also able to revise a decision by navigating back to a previous task [30].

The users are given feedback about the purpose of each task and the users can see at all times

where they are in the sequence and which steps are parts of the sequence. When the complex task is completed, feedback is provided to show the user that the tasks have been completed and optionally results have been processed.

Users that know the default options can immediately use a shortcut that allows all the steps to be done in one action. At any point in the sequence it is possible to abort the task by choosing the visible exit [30].

It is important to note that the navigation buttons suggest to the users that they are navigating a path with steps. Each task is presented in a consistent fashion enforcing the idea that several steps are taken. The task sequence informs the user at once which steps will need to be taken and where the user currently is. The learn ability and memor-ability of the task are improved but it may have a negative effect of the performance time of the task. When users are forced to follow the order of

tasks, users are less likely to miss important things and will hence make fewer errors [30].

#### **Known Uses**

1. **CAPIT [19]:** An ITS that teaches the rules of English capitalization and punctuation. The main interface is illustrated in figure (4-13); instructions relevant to the current problem are clearly displayed at the top of the screen. Immediately below the instructions, and clearly highlighted, is the current problem. There are also navigation buttons to move back or next to assist student in navigating without overwhelming him with plenty of lists or options. Motivation is provided in two forms. Firstly, students accrue points every time they submit a correct solution. The total number of points accrued so far, and the value in points of the current problem, is clearly displayed on the main interface. Secondly, when a correct solution is submitted, a simple animation is displayed as a reward.

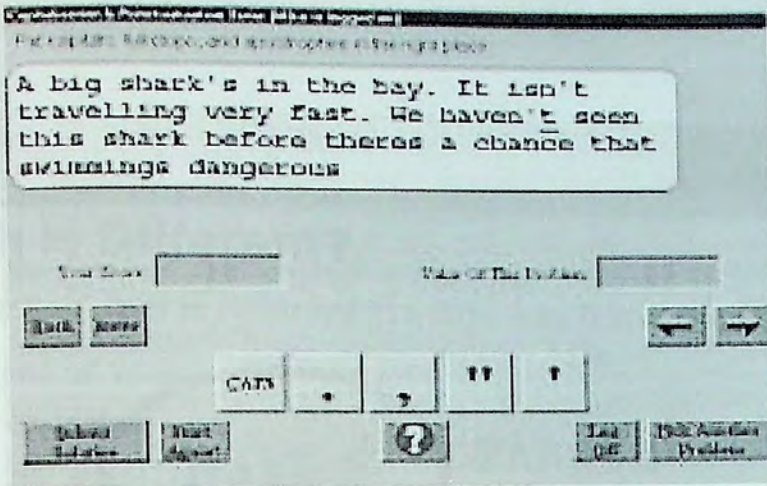


Figure (4-13): CAPIT Main User Interface [19]

2. **The German Tutor [15]:** An intelligent tutoring system for teaching German. After the student finishes answering the current question he can go on to the next exercise with the "Weiter" (next) button. The German Tutor utilizes the **Wizard Pattern** by using navigation buttons to move to next exercise to assist students in navigating without overwhelming them with plenty of lists or options. Figure (4-14) illustrates the **Wizard Pattern** in the context of the dictation exercise in the German Tutor.

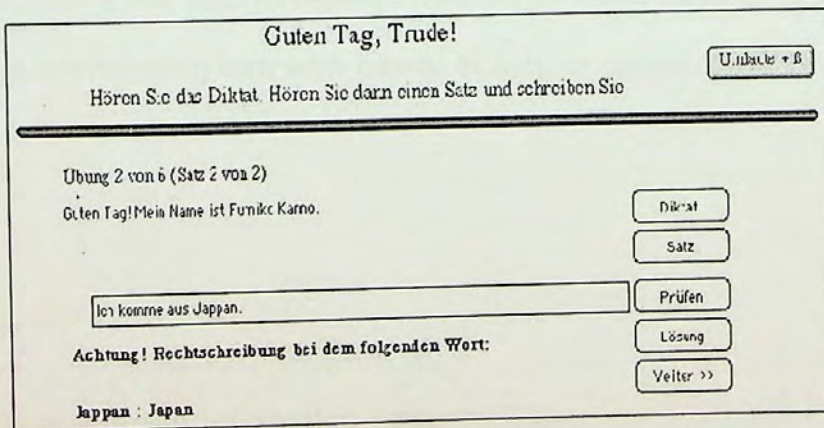


Figure (4-14): Dictation Exercise in the German Tutor [15]

3. **The Greek Tutor [26]:** An intelligent tutoring system for teaching Greek. As shown from the example in figure (4-15), this exercise displays a number of words all except one of which belong to the same category. The student task is to identify the one that differs from the others. The divergent word may differ syntactically, semantically or

pragmatically from the remaining words. An example of this exercise type is given in the figure (4-15):

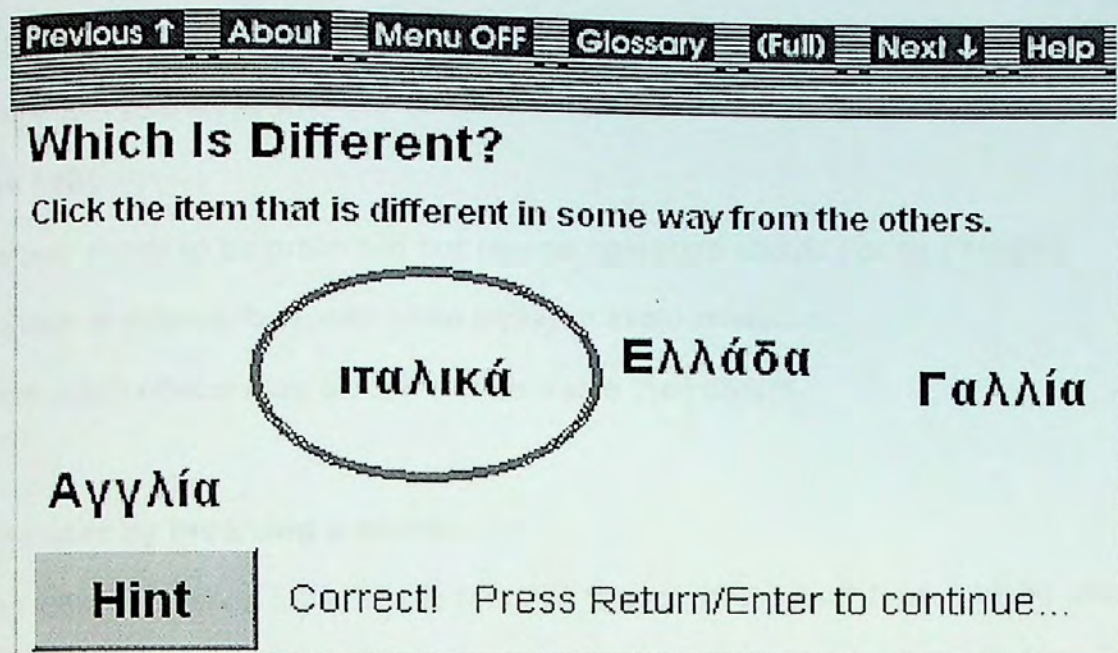


Figure (4-15): Which Word is Different in the Greek Tutor [26]

As illustrated in the previous figure instructions relevant to the current problem are clearly displayed at the top of the screen. Immediately below the instructions, and clearly highlighted, is the current problem. There are also navigation buttons to move back or next to assist student in navigating without overwhelming him with plenty of lists or options.

#### Related Patterns

User Goals

Shield

#### References

The **Wizard Pattern** is one of the Interaction Patterns [30].

#### 4.3.4.4 Pattern 18: Shield

##### Classification

User Interface Patterns (Interaction Patterns) [30].

##### Problem

The user may accidentally select a function that has irreversible (side) effects or require a lot of resources to undo/reverse. The (side) effects may lead to unsafe or highly undesired situations.



Often it is the combination of action arguments that makes it severe, and the user may not be aware of this, since it is normally safe [30].

### Usability Principle

Error Management (Safety) [30].

### Motivation [30]

- The user needs to be protected but normal operation should not be changed.
- The user is striving for speed while trying to avoid mistakes.
- Some (side) effects may be more undesirable than others.

### Solution

**Protect the user by inserting a shield.**

1. Add an extra protection layer to the function to protect the user from making mistakes. The user is asked to confirm his intent with the default answer being the safe option. The extra layer causes the user to require 2 repetitive mistakes instead of 1. The safe default decreases the chances for a second mistake. The solution increases safety, reduces errors and increases satisfaction. However, it requires extra user action which leads to lower performance time [30].
2. Prevent the user from being able to make an action that has irreversible (side) effect by disabling the action that will lead him to this situation.

### Known Uses

1. **The German Tutor [15]:** An intelligent tutoring system for teaching German. In the event of error the student can either correct the error and resubmit the sentence by clicking the "Prüfen" (check) button, or peek at the correct answer(s) with the "Lösung" (answer) button, then he can go on to the next exercise with the "Weiter" (next) button. If the student chooses to correct the sentence it will be checked for further errors.

It is important to note that the German Tutor is utilizing the **Shield Pattern** by disabling the "Weiter" (next) button until the student finishes the current question either correctly or by knowing the right answer through the system.

## Related Patterns

Course Creation and Customization

User Goals

Whole Part

Wizard

## References

The **Shield Pattern** is one of the Interaction Patterns [30].

### 4.3.4.5 Pattern 19: Warning

#### Classification

User Interface Patterns (Interaction Patterns) [30].

#### Problem

The user may unintentionally cause a problem situation which needs to be resolved [30].

#### Usability Principle

Error prevention (Safety) [30].

#### Motivation [30]

- Work may be lost if the action is fully completed.
- The system can or should not automatically resolve this situation so the user needs to be consulted.
- Some actions are difficult or impossible to recover from.
- Users may not understand why an action could be damaging.
- Users may not understand the consequences or options.

#### Solution

Warn the user before continuing the task and give the user the chance to abort the tasks.

#### The warning should contain the following elements [30]:

- A summary of the problem.
- The condition that has triggered the warning.
- A question asking the users whether to continue the action or take on other actions.
- Two main choices for the user, an affirmative choice and a choice to abort.

In some cases there may be more than two choices. Increasing the number of choices may be acceptable in some cases but strive to minimize the number of choices [30].

By stating the triggering condition the user can understand why the warning appears. Once that is understood the question leaves the user with only two options. By providing only two options the choice is made simple for the user: continue or abort. More options make it increasingly difficult for the user to make a decision. The solution decreases errors and increases satisfaction [30].

### **Related Patterns**

Course Creation and Customization

Wizard

### **References**

The **Warning Pattern** is one of the Interaction Patterns [30].

## **4.4 PLITS: A Pattern language for Intelligent Tutoring Systems**

The design and implementation of ITSS is a very complex task that hampers the development process, since it incorporates a variety of organizational, administrative, instructional and technological components. Therefore systematic, disciplined approaches must be devised in order to leverage the complexity and assortment of ITS and achieve overall product quality within specific time and budget limits [4].

In this research we tried to mine patterns by reverse-engineering the systems that embed good designs in order to make that design explicit, and be able to communicate it to other designers, so that it becomes common practice. As a result, good design experience will be codified and a more systematic development process for these systems will be provided. The proposed pattern language uses patterns that are semantically organized and categorized, and form the basic core of a pattern language for Intelligent Tutoring Systems. This research aims take research steps in that direction by proposing a pattern language for intelligent tutoring systems. In this way, designers of new or existing ITS, especially inexperienced designers, can take advantage of

previous design expertise and save precious time and resources.

There is only one Pattern Language that discusses the common patterns in ITS Architectures [32]. However; up to our knowledge there is no pattern language for ITS implementation.

In section 4.3 we described the patterns that we found useful for ITS developers and designers.

In this section we will rediscuss these patterns from a different perspective.

Table (9) maps our proposed ITS functional requirements to the corresponding ITS patterns.

Each pattern is described using the following information:

- Pattern category
- Pattern Name
- The Functional Requirement that this pattern offers in ITS context
- The ITS module that the pattern belongs to
- Pattern Source

This summarized description can be used as a roadmap for ITS developers and designers.



Table (9): Mapping Between the Functional Requirements and the Corresponding ITS Patterns

#	Pattern Category	Pattern Name	Functional Requirement	ITS Module	Source
1	Pedagogical Patterns	Early Bird	Preparing and Organizing Curriculum	Domain Module	[31]
2	Design Pattern	Whole Part	o Curriculum Composition	Domain Module	[12]
			o Lesson Presentation Planning o Exam Generation	Tutor Module	
3	Structural Patterns	New Flyweight	Curriculum Instantiation	Domain Module	[7]
			Arabic Alphabetic Letters and Shapes Instantiation		
4	Creational Patterns	Builder	Curriculum Construction and Loading	Domain Module	[7]
5	Creational Pattern	Singleton	Curriculum Instantiation	Domain Module	[7]
			Arabic Alphabetic Letters Instantiation		
			Collective Student Information Instantiation	Tutor Module	
			User Interface Instantiation	GUI Module	
6	Access Patterns	Registration-Authentication-Access-Control	Students Registration and Access Control	Student Model	[35]
7	Adaptive Patterns	User Model Definition	Student Model Standardization	Student Model	[28]
8	Adaptive Patterns	User Goals	Determining and Achieving Student Goals	Student Model	[28]
9	Adaptive Patterns	New Student Model Initialization	Initializing Student Model	Student Model	[28]
10	Adaptive Patterns	New Student Model Maintenance	Update and Maintenance of Student Model	Student Model	[28]
11	Structural Patterns	New Adapter	Representing Student Status Per Topic	Student Model	[7]
			Representing Student Status Per Exercise		
12	Behavioral Patterns	New Strategy	Sorting Topics, Learn Items, Examples and Exercises	Tutor Module	[7]
13	Behavioral Pattern	Template Method	Question Title Display	Tutor Module	[7]
14	Design Pattern	Master Slave	Student Exam Grade Computation	Tutor Module	[12]
15	Behavioral Patterns	Observer	Monitoring Student Progress and Updating Students' Progress Reports	Tutor Module	[7]
16	Instructional Patterns	Course Creation and Customization	Creating and Customizing Curriculum	GUI Module	[35]
17	Interaction Patterns	Wizard	Guiding Students Through a Complex Task	GUI Module	[30]
18	Interaction Patterns	Shield	Preventing Students from Making Mistakes that has Side Effects including:	GUI Module	[30]
			Preventing Students from not choosing topic(s) of interest		
			Preventing Students from choosing a topic without choosing its prerequisites as well		
			Preventing Students from not answering a Question		
19	Interaction Patterns	Warning	Warning Students from Making an Action that May Lead to Problems.	GUI Module	[30]

Figure(4-16) represents an overview of the Pattern Language Approach. This Pattern Language approach will be explained in detail in chapter 5 where we will show how we used **PLITS** to implement the **Arabic Tutor**, a web based intelligent tutoring system for teaching a subset of the Arabic Language.

In chapter 4 we tried to list patterns inside each ITS module and in chapter 5 we will identify the recommended sequence of using the identified patterns to implement the **Arabic Tutor**.

In figure (4-16) we used two types of arrows:

	<p>The dotted arrow is used to indicate the occurrence of a relationship between the ITS Modules.</p>
	<p>The solid arrow is used to indicate the occurrence of a relationship between patterns. This relationship is either a precedence or dependency as will be illustrated in more details in chapter 5.</p>

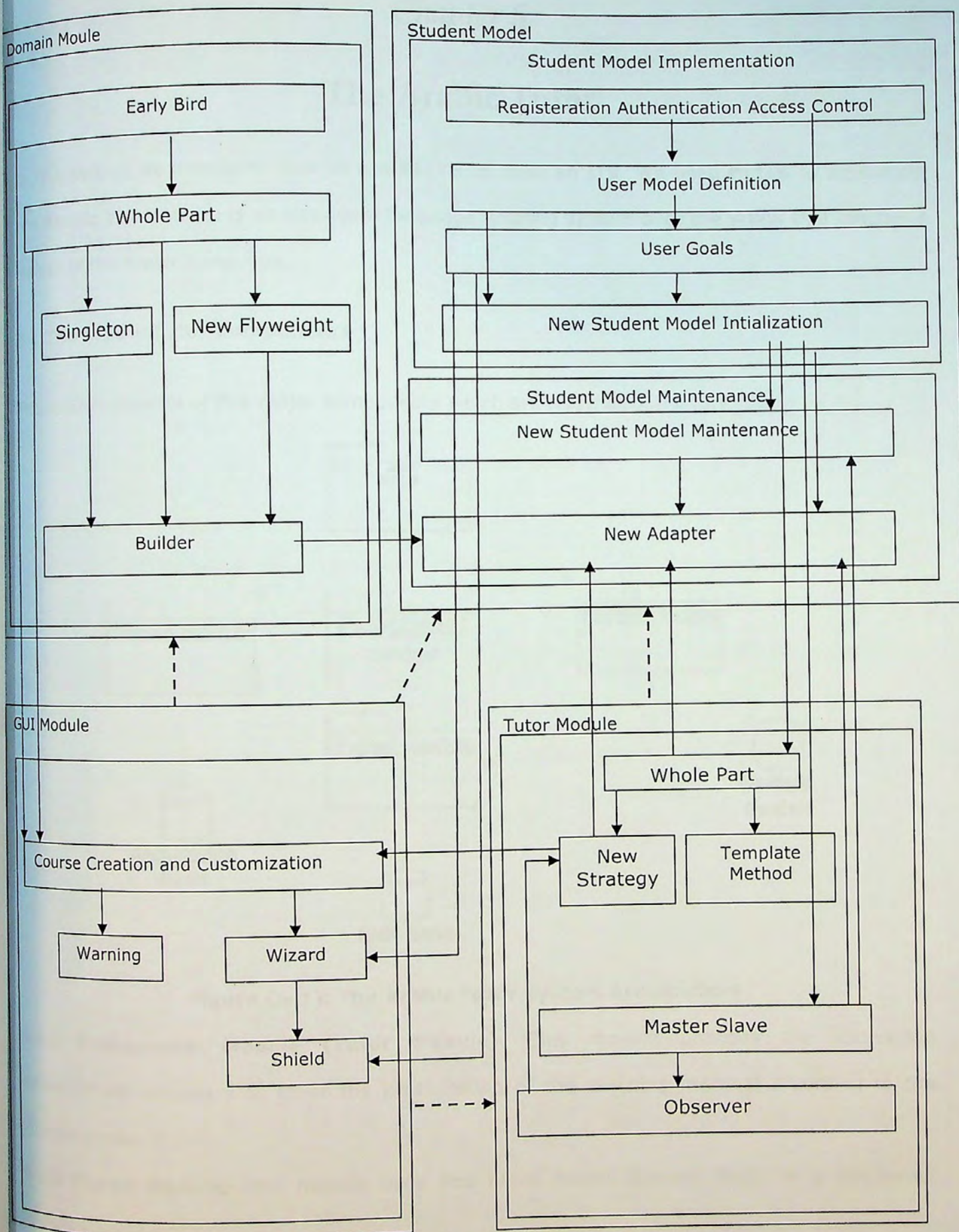


Figure (4-16): PLITS: Pattern Language Approach

## Chapter 5

# The Arabic Tutor

In this section we illustrated how to use **PLITS** to build an ITS. We used **PLITS** to implement the **Arabic Tutor** which is an intelligent language tutoring system over the WWW that teaches a subset of the Arabic Language.

### 5.1 The Arabic Tutor Components

The system consists of five major components which are illustrated in figure (5-1):

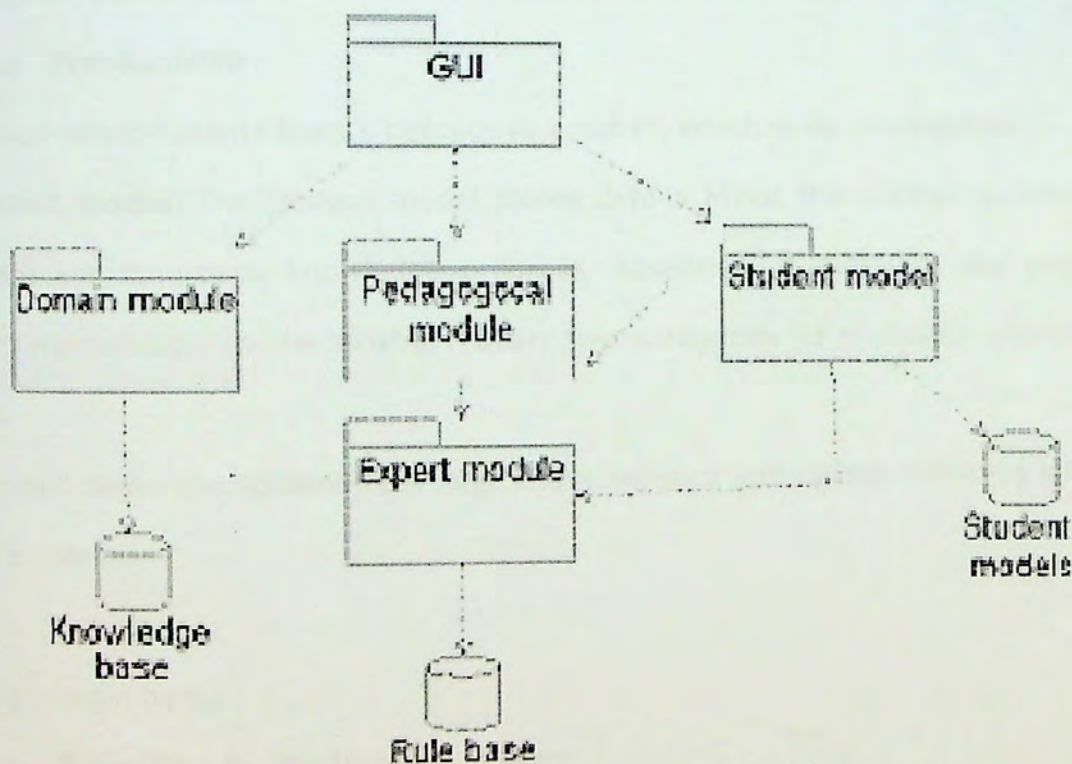


Figure (5-1): The Arabic Tutor System Architecture

**5.1.1 Pedagogical Module (Tutor Module):** This module provides the knowledge infrastructure necessary to tailor the presentation of the teaching material according to the student model.

**5.1.2 Expert Module:** This module uses Jess (Java Expert System Shell) as a rule-based inference engine.



**5.1.3 Domain Module:** This model contains the knowledge about the actual teaching material. The domain knowledge of the **Arabic Tutor** is represented using a semantic network depicting the relations between the concepts of the domain. Each node in the domain knowledge represents a certain category of concept, which may be further divided into smaller sub-concepts. There are three kinds of links between nodes:

- o **Part-of**

A part-of relation points from a more general to a more specific concept, which is one of its parts.

- o **Is-a**

An is-a relation, points from an instance of a concept to the concept.

- o **Prerequisite**

A precondition relation points from a concept to another, which is its prerequisite.

**5.1.4 Student Model:** The Student model stores details about the student's current problem-solving state and long-term knowledge progress, essential for adapting the material to the student's characteristics. In the **Arabic Tutor**, two categories of student's characteristics are considered:

- Personal data - the student's personal characteristics include the following information:
  - o Name
  - o E-mail
  - o Login name
  - o Topics that he is interested to learn
- Performance data - the current level of mastery of learning material.

#### **Components of the Student Model**

Several components in the Pedagogical module make decisions about the student model. The decisions are related to the selection of an appropriate topic for the student, curriculum sequencing, and so on. During such processes, the Pedagogical module communicates with the student model in order to get relevant information.

**The student model has two components:**

- The Performance model which stores data related to the assessment of the student's overall skills, as well as data related to the student's previous knowledge, etc.
- The Teaching history model which keeps track of the material presented to the student during the session and the student's mastery of teaching units.

**5.1.5 GUI Module:** The GUI Module of the **Arabic Tutor** is divided into two parts:

○ **Student Interface**

This consists of a set of dynamically constructed HTML pages and forms. The decision about the content and the form of each page is made by the tutoring component. The student interface will be illustrated later in this chapter.

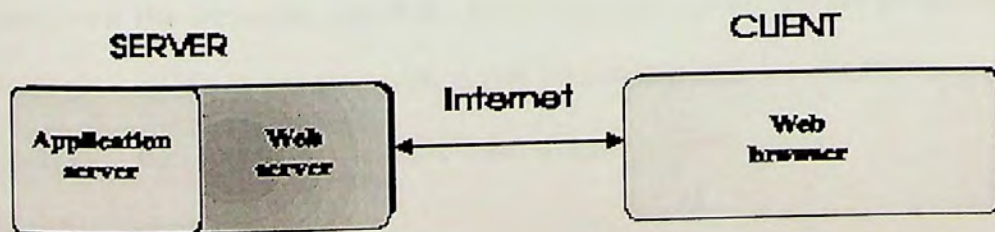
○ **Teacher Interface**

This part represents the interface that is viewed by the teachers. This interface provides the teacher with the following:

1. Easy interface to create teaching material. The teaching material represents the domain knowledge that will be used by the tutor module later on.
2. Access to Students' Progress Reports.

**5.2 The Arabic Tutor Architecture**

The **Arabic Tutor** is an intelligent tutoring system over the World Wide Web and is based on the client server architecture. The architecture adopted is the centralized architecture illustrated in figure (5-2):



**Figure (5-2): Arabic Tutor Centralized Web Architecture [5]**

It consists of a Web server, an application server and student interface. The application server and the Web server run on the server side, while the student interface is displayed in a Web

browser on the client's machine. The application server performs all tutoring functions. The interface consists of a set of HTML pages. The student interacts with HTML entry forms, and the information is sent to the Web server, which passes the student's requests and actions to the application server[5].

**The centralized architecture offers a number of advantages [5]:**

- Students need only a web browser to use our system.
- Students are not required to download any software in order to start using the system.
- Any updates or improvements to the system are automatically propagated to students without the need to download the updates once more.
- All student models are kept in one place (on the server) and the student can use the system from any machine.

**Disadvantages of the centralized architecture:**

- **Reliance on the server**

It is absolutely vital that the network server is robust and very reliable. If the network server fails on a thin client network, all terminals on the network are unusable until the network is restored. On a traditional server based network, only access to network resources (shared peripherals devices, shared files) is lost in this event so applications on an individual machine will still continue to work when the network goes down.

- **Bandwidth issues**

Like any network, the amount of available bandwidth for any user will depend on the number of other users on the network. As thin client requires practically all processing to be carried out on the server, there is considerably more network traffic as all processing commands and resulting actions need to be transmitted across the network.

### **5.3 The Arabic Tutor Features**

- 1) Curriculum Sequencing: Curriculum sequencing provides the student with the most suitable individually planned sequence of knowledge units to learn and sequence of learning tasks (examples, exercises, etc.) to work with. In other words, it helps the student to find an "optimal path" through the learning material.

**There are two kinds of curriculum sequencing techniques.**

- High-level sequencing or knowledge sequencing determines next concept or topic to be taught.
  - Low-level sequencing or task sequencing determines next learning task (problem, example, and test) within current topic.
- 2) The system displays only one error message at a time, and the student is expected to correct the error (and possibly any others) and resubmit the problem before any more feedback is displayed.
  - 3) Error specific feedback that differs in its level of detail according to student knowledge level per addressed topic.
  - 4) The system has two interfaces one for the students who need to learn a certain material and to take exams. And the other is for teachers who want to monitor student progress per topic.

Knowledge for the tutor was primarily acquired from a number of text books from the American University in Cairo library. We also had considerable input from a professional teacher, at the Arabic Language Institute in the American University in Cairo who made useful suggestions as to which topics, learn items, types of questions and examples rules the tutor should cover and in what sequence. For example, most common errors that students are likely to make when they study the suggested topic. Knowledge of common errors like these is more difficult to find in textbooks, and so the input from the teacher was invaluable. He also helped us in tailoring the appropriateness and vocabulary of the system's feedback, explanatory and introductory messages to the students.

#### **5.4 Building the Arabic Tutor Using PLITS**

In this section we will illustrate using **PLITS** to create the **Arabic Tutor**. As we mentioned earlier the **Arabic Tutor** is composed of four modules:

- **Pedagogical Module (Tutor Module)**
- **Expert Module**
- **Domain Module**

- **Student Model**

In chapter 4 we discussed the proposed patterns that **PLITS** is composed of however, in this section we will list them according to their recommended usage sequence in order to reach a final product represented by the **Arabic Tutor**.

Please refer to appendix F for a detailed class diagram for all design patterns included in **PLITS**.

#### 5.4.1 Early Bird Pattern in the Arabic Tutor

Any ITS is actually based on teaching material that represents the domain module and that needs to be conveyed to students. Students need to be able to separate the key concepts from the details that support them. They need to see where they are headed. Therefore, we needed to begin with the **Early Bird Pattern** to help us in preparing and organizing the course so as to ensure that the most important topics are taught first. [31].

The problem that we faced when we tried to accomplish our goal was that there are many interrelated topics. We succeeded in identifying the big ideas through a number of steps:

1. We depended on the help of an Arabic professor, Mr. Abbas Tonsi, Director of the Computer Aided Language Learning Department in the Arabic Language Institute in the American University in Cairo. We benefited from his expertise in determining the big ideas and introduced them first.
2. We used a number of Arabic text books that are specialized in teaching Arabic language to both native speakers and foreigners.
3. We implemented a teacher interface in the **Arabic tutor** that provides the teacher with easy interface to create his/her teaching curricula including topics, learn items, examples and exercises .In addition, we allowed the teacher to enter the sequence number for each of them to make sure that big ideas will be introduced first. This teacher interface will be introduced in more details in the **Course Creation and Customization Pattern**.

4. We introduced the big ideas and their relationships at the beginning of the course and returned to them repeatedly throughout the course. This will be illustrated in more detail later in this chapter.

#### 5.4.2 Whole Part Pattern in the Arabic Tutor

After we used the **Early Bird Pattern in Preparing and Organizing Curricula** we needed to compose the teaching material and all its constituent parts. The **Whole Part Pattern** helps in representing objects that are composed of other objects. Such aggregate objects form units that are more than just a mere collection of their parts [12]

**Typical situations in which the Whole Part Pattern can be applied in ITS design:**

- **Composing Curricula**
- **Lesson Presentation Planning**
- **Exam Generation**

**In the Arabic Tutor we used Whole Part Pattern in:**

#### First Application of the Whole Part Pattern in the Arabic Tutor: Composing Curriculum

In order to represent the aggregation relationship between the following items:

- Teaching Material is composed of a number of topics.
- Each Topic is composed of a number of learn items.
- Each Topic has a number of exercises related to it.
- Each Learn Item has a number of examples related to it.

The structure of the **Whole Part pattern** in the **Arabic Tutor** is represented in figure (5-3):

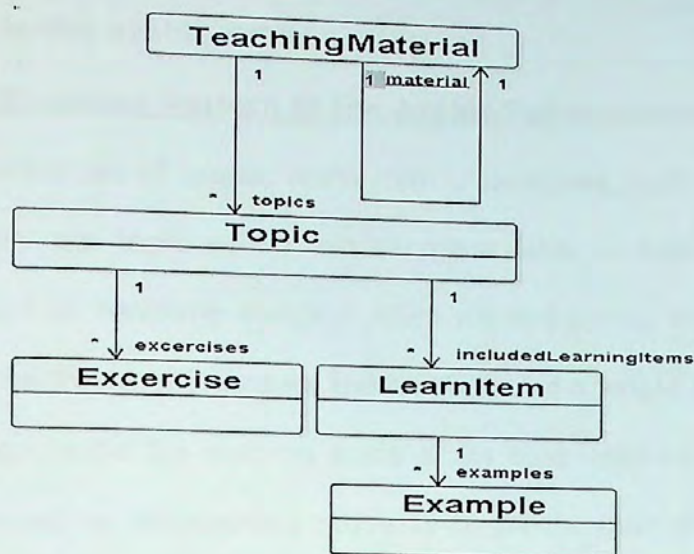


Figure (5-3): Whole Part Pattern Structure in the Arabic Tutor

As illustrated in figure (5-3), we have two participants:

- **Topic Class (Whole Class):** that represents a complex object and acts as a container for other primitive objects (learn items).
- **LearnItem Class (Part Class):** that represents the primitive elements.

We took into consideration to Declare management operation that manages the LearnItem objects like addNewItem (), deleteItem (), findItem () and loadItems () in the Topic class to make these management operations available only in Topic Class and not available in LearnItem Class.

In addition we have a second occurrence for the Whole Part Pattern with the following participants:

- **Topic Class (Whole Class):** that represents a complex object and acts as a container for other primitive objects (Exercise).
- **Exercise Class (Part Class):** that represents the primitive elements.

Moreover, we have a third occurrence for the Whole Part Pattern with the following participants:

- **LearnItem Class (Whole Class):** that represents a complex object and acts as a container for other primitive objects (Example).
- **Example Class (Part Class):** that represents the primitive elements.

### 5.4.3 Singleton Pattern in the Arabic Tutor

#### First Application of the Singleton Pattern in the Arabic Tutor: Curriculum Instantiation

Each student views a different set of topics, learn items, exercises, and examples according to his current knowledge level per topic as we will illustrate later in this chapter. However, all students use the same pool of Teaching Material, thus we needed to implement the Teaching Material using the **Singleton Pattern** to ensure the existence of a single instance by making the TeachingMaterial Class responsible for keeping track of its sole instance and ensuring that no other instance can be created by intercepting requests to create new objects, and providing a way to access the instance.

The Structure of the TeachingMaterial class is illustrated in figure (5-4):

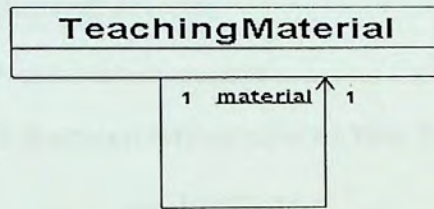


Figure (5-4): Singleton Pattern in the TeachingMaterial Class in the Arabic Tutor

### 5.4.4 New Flyweight Pattern in the Arabic Tutor

#### First Application of the New Flyweight Pattern in the Arabic Tutor: Curriculum

##### Instantiation

Moreover, the **Arabic Tutor** application is initialized by loading all the needed Teaching Material objects (Topic, LearnItem, Exercise and Example) that represent the domain module. We needed to instantiate Teaching material and at the same time not to allow students to instantiate this shared object in order to avoid duplication. Thus we used the **New Flyweight Pattern** which uses sharing to support large numbers of fine grained objects efficiently in order to allow their use at fine granularities without prohibitive cost. Flyweights model concepts or entities that are normally too plentiful to represent with objects [7].



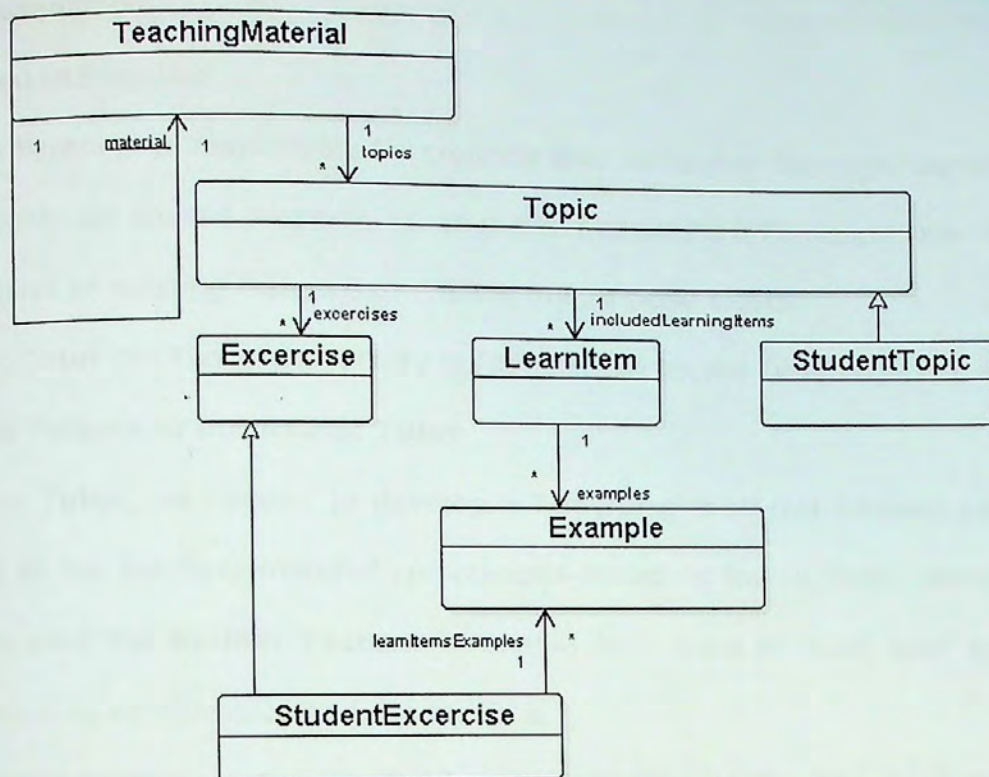


Figure (5-5): New Flyweight Pattern Structure in the TeachingMaterial Class in the Arabic Tutor

**New Flyweight Participants in the Arabic Tutor:**

1. **ConcreteFlyweight** is a sharable object. Any state it stores must be intrinsic; that is, it must be independent of the ConcreteFlyweight object's context.

In the **Arabic Tutor** domain Concrete Flyweight is represented in the domain of ILTS by an instance of each of the following classes:

- Example
- Exercise
- Topic
- LearnItem

2. **Unshared ConcreteFlyweight:** It's common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the Flyweight object structure.

In the **Arabic Tutor** Unshared ConcreteFlyweight is represented by an instance of each of the following classes that will be explained in more detail when we discuss the **New Adapter Pattern**:

- StudentTopic
- StudentExercise

**3. Flyweight Factory:** Is responsible for creating and managing Flyweight objects and Ensuring that Flyweights are shared properly. When a client requests a Flyweight, the FlyweightFactory object supplies an existing instance or creates one, if none exists.

In the **Arabic Tutor** the Flyweight Factory is represented by the TeachingMaterial class.

#### 5.4.5 Builder Pattern in the Arabic Tutor

In the **Arabic Tutor**, we needed to develop a **Teaching Material Loader and Constructor** that can load all the teaching material components including topics, learn items, examples and exercises. We used the **Builder Pattern** to enable each class to build itself and afterwards it should start building components or parts inside it.

**We followed the following steps in the implementation of the Teaching Material:**

1. Created a Loader interface.
2. Created a TeachingMaterial object: This object instantiates a Topic object corresponding to every Topic and then sends a load () message to that object; this Load () message is responsible for building all object components or parts.
3. The Topic object then instantiates a LearnItem object corresponding to each learn item and sends a load () message to that object.
4. The Topic object then instantiates an Exercise object corresponding to every exercise and sends a load () message to that object.
5. And finally, the LearnItem object instantiates an Example object that corresponds to every example and sends a load () message to that object.

It is illustrated from the previous steps that the build responsibility is divided between TeachingMaterial, Topic and LearnItem. Every class is a builder and a director at the same time except the first one (TeachingMaterial) which acts as a director only because it directs another class (Topics) to build itself and the (Exercise class ) which acts as a builder only for itself and it doesn't direct any other class to build itself.

The structure of the Builder Pattern in the Arabic Tutor is illustrated in figure (5-6):

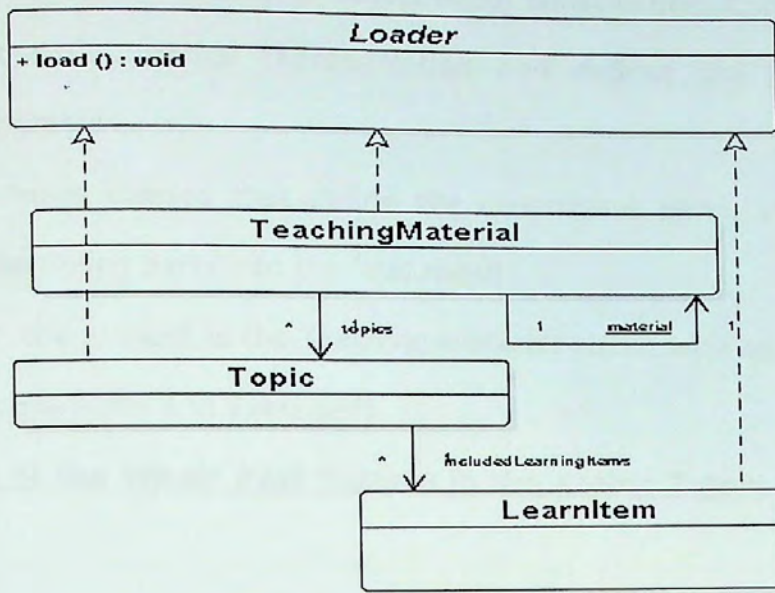


Figure (5-6): Builder Pattern Structure in the Arabic Tutor

**Builder Participants in the Arabic Tutor:**

- **Builder**
  - Specifies an abstract interface for creating parts of a Product object and is represented by the Loader interface.
- **ConcreteBuilder [7]**
  - Constructs and assembles parts of the product by implementing the Builder interface.
  - Defines and keeps track of the representation it creates.
  - Provides an interface for retrieving the product.

The LearnItem class and the Topic class act as our ConcreteBuilder.

- **Director**
  - Constructs an object using the Builder interface and is represented by the Loader interface.

Thus it can be shown that we have one interface that acts as a builder and a director at the same time.

• **Product**

- Represents the complex object under construction. ConcreteBuilder builds the product's internal representation and defines the process by which it's assembled.
- Includes classes that define the constituent parts, including interfaces for assembling parts into the final result.

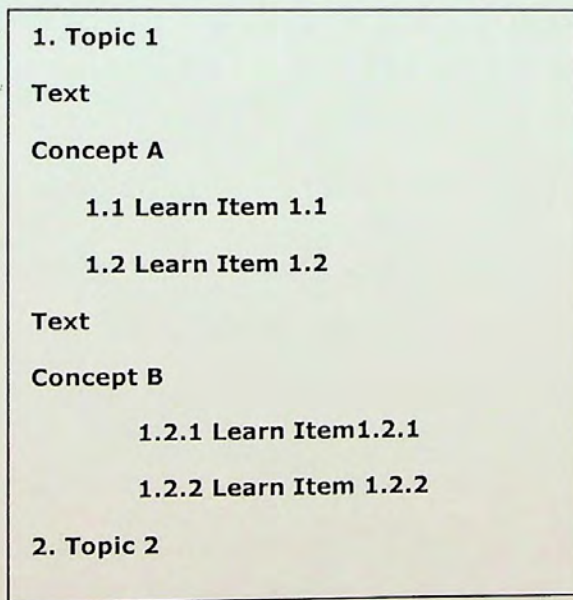
In the **Arabic Tutor**, the product is the Teaching material object with all its constituent objects (Topics, Learn Items, Examples and Exercises)

**Second Application of the Whole Part Pattern in the Arabic Tutor: Lesson Presentation**

**Planner:**

We needed to build an agenda of the contents to be presented during the lesson. Each student may need to learn either a complex object (Topic; which can be divided into simple elements represented in a number of Learn Items) or a primitive object (Learn Item).

This will produce an agenda that is illustrated in figure (5-7):



**Figure (5-7): Student Learning Agenda in the Arabic Tutor**

**5.4.6 Registration-Authentication-Access Control Pattern in the Arabic Tutor**

ITSs are large, multi-user systems. Due to security, privacy and institutional policy reasons, access to the resources of the ITS must be restricted to authorized users only. Additionally, user roles vary and can be divided into two major roles:

- **Teacher Roles:** Teachers access the ITS to add teaching material including new topics, learn items, examples and exercises .In addition, they access the ITS to monitor current knowledge level for each student through students' progress reports.
- **Student Roles:** Students access the ITS in order to learn a certain course and benefit from the intelligence and adaptivity of the ITS.

Consequently, systems must assign specific rights to the various systems resources according to the role of each user. In the **Arabic Tutor** we used the **Registration-Authentication-Access Control Pattern** to provide a standard registration mechanism for every user of the system through a web interface that is illustrated as follows:

The student will first sign up in our **Arabic Tutor** through the following screen (figure (5-8)):

**Figure (5-8): Student Signup Form in the Arabic Tutor (\*)**

Names are abbreviated to protect student's privacy

#### 5.4.7 User Model Definition Pattern in the Arabic Tutor

A student model is essentially the image the system has about the learner. The closer the student model is to the learner's real characteristics and needs, the better the personalization. Therefore, the information kept in the student model must describe the learner in the best way

possible, but at the same time allows the model to be flexible in its manipulation. We need a student model that is small, compact and flexible.

#### 5.4.8 User Goals Pattern in the Arabic Tutor

In the **Arabic Tutor**, we used the **User Model Definition Pattern** and tried to implement an effective user model definition that is comprised of the following elements:

##### First element in User Goals Pattern: Student goals

Student goals are related to the learning goals e.g. "to complete course X". As shown in figure (5-8); students are required to determine their long term goals from using the **Arabic Tutor** in the sign up form. This is achieved with the help of the **User Goals Pattern**.

We need to include specific student goals in the student model in order to facilitate adaptation, and capture the real intent of the learner with respect to the learning material.

**The student goals can be divided into two categories:**

- **Long-Term Goals:** Educational goals that are valid for a longer period of time and require significant effort to be met. Long-term goals are usually determined by the learners.

##### **Long Term Goal Handling in the Arabic Tutor:**

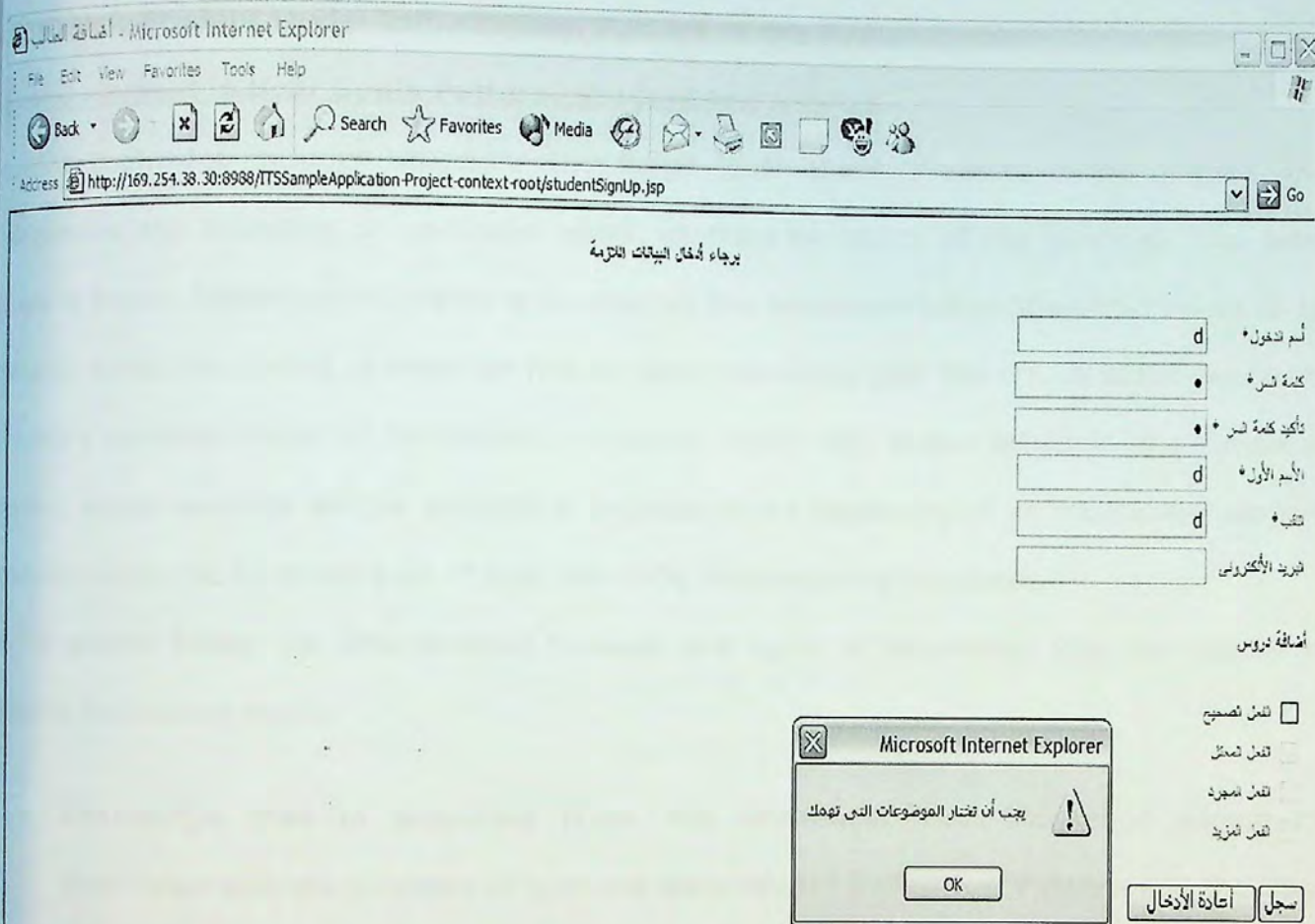
As illustrated in figure (5-8) students who use the **Arabic Tutor** are required to determine their goal from using the system. This is achieved by allowing the students to choose the topics that are of interest to them. These topics may be dependent on each other. Each topic may have successors, predecessors, or topics that can be taught in parallel with this topic. The user may accidentally choose a topic that he is interested to learn without choosing its necessary prerequisites. Thus he may start learning a certain topic without acquiring the necessary base to fully understand this topic.

#### 5.4.9 Shield Pattern in the Arabic Tutor

**First Application of the Shield Pattern in the Arabic Tutor: Preventing Students from not choosing topic(s) of interest:**

The **Arabic Tutor** utilizes the **Shield Pattern** to prevent the user from being able to make an action that has irreversible side effects by disabling the action that will lead him to this situation.

It is also the responsibility of the **Arabic Tutor** to make sure that the student chooses one or more topics that he is interested in learning, if not the student will be prevented from proceeding and he will get the errors message shown in figure (5-9):



**Figure (5-9): Student Aborted Attempt to Start Learning Without Determining his Long Term Goals**

Names are abbreviated to protect student's privacy

**Second Application of the Shield Pattern in the Arabic Tutor: Preventing Students from choosing a topic without choosing its prerequisites as well**

It is also the responsibility of the **Arabic Tutor** to check that the student cannot choose a topic that has a certain prerequisite without choosing all its prerequisites as well. In order to achieve this we utilized the **Shield Pattern**. In the **Arabic Tutor**, we prohibit the student from making such a mistake by disabling any topic that has prerequisites and we don't enable this topic until the student chooses all its prerequisites as well. Thus we prevent the student from being able to make mistakes in the first place and we create a shield against harmful actions.

- **Short Term Goals:** Educational goals that are valid for a shorter period of time and require relatively moderate effort to be met. Short term goals are usually determined by the **Tutor Module** component of the **Arabic Tutor**.

#### 5.4.10 New Student Model Initialization Pattern in the Arabic Tutor

##### Second element in User Goals Pattern: Student knowledge

Student knowledge includes student's knowledge level about concepts to be learned and weaknesses and strengths on particular areas, sections or points of the concepts. The **New Student Model Initialization Pattern** focuses on the necessary information that needs to be acquired about the student in order for him to start interacting with the ITS. It is not necessary to have a complete model of the student; a partial model with proper selection of a subset of student model elements will be acceptable because in the beginning of an interaction session, students do not like to spend a lot of time providing information about them.

In the **Arabic Tutor**, we differentiated between two types of knowledge that are needed to initialize the student model:

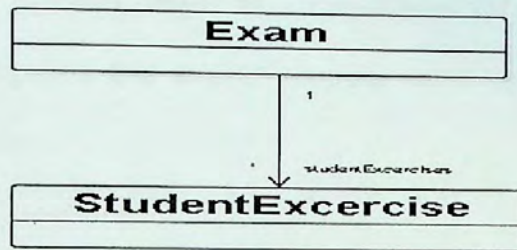
- **Knowledge that is acquired from the students:** This knowledge should be determined with the guidance of both the **User Model Definition Pattern** and the **User Goals Pattern**.
- **Knowledge that can be acquired automatically by the system:** This knowledge can be acquired with the help of the **New Student Model Initialization Pattern** by implementing an **Entry Exam** that uses an **Exam Generator Component** to measure student knowledge level. This component uses the pool of questions that was filled earlier by the teachers and randomly chooses a set of questions that represent different difficulty levels thus can accurately measure current student knowledge level. The entry exam result will be used to initialize the student model with the current knowledge level of the student.



### Third Application of the Whole Part Pattern in the Arabic Tutor: Exam Generation

This exam generator component utilizes the **Whole Part Pattern**. The **Whole Part Pattern** is used to represent aggregate objects that are more than just a mere collection of their parts. In addition it is used to represent a hierarchical relationship between objects [12]. Each Exam is composed of a number of exercises.

The structure of the **Whole Part Pattern** in the Entry Exam of the **Arabic Tutor** is represented in figure (5-10):



**Figure (5-10): Whole Part Pattern Structure in the Arabic Tutor (Exam-StudentExercises)**

As illustrated in figure (5-10), we have two participants:

- **Exam Class (Whole Class):** Represents a complex object and acts as a container for other primitive objects (StudentExercises). This class is responsible for choosing, organizing questions and computing student score.
- **StudentExercise Class (Part Class):** Represents the primitive elements. This class is responsible for determining the right answer, student answer, exercise difficulty level and student score per exercise.

Thus we benefited from the **Whole Part Pattern** in the separation of concerns between the Exam and the StudentExcercise Class. It therefore becomes easier to implement complex strategies by composing them from simpler services than to implement them as monolithic units.

### Arabic Tutor Exercise Types:

- Enter a verb
  - The enter a verb exercises are considered to be one of the strongest features in the **Arabic Tutor**. Students will provide natural language input rather than selecting exclusively from among pre-defined answers. Students are presented with an exercise that asks them to type a verb with specific features. The student entry is passed to the **Arabic Tutor expert module** that uses Jess (Java Expert System Shell) as a rule-based inference engine. Student's entry is checked against the expert module rules to verify the correctness of the student answer.
- Multiple Choice Questions
  - Choose one answer
  - Choose two answers
  - Choose all that apply
- True/ False

#### 5.4.11 Template Method Pattern in the Arabic Tutor

In order to automatically handle question title display in the **Arabic Tutor** we used the **Template Method Pattern** that defines the skeleton of an algorithm in an operation, deferring some steps to subclasses. Template Method subclasses redefine certain steps of an algorithm without changing the algorithm's structure. Each type of question has a certain title that needs to be displayed at runtime; we used the **Template Method Pattern** to allow for the automatic display of this title at runtime as illustrated in table (10).

**Table (10): Implementation of Question Type in the Template Method.**

Question Type	Question Subtype	Question title displayed
Multiple choice questions	○ Choose one right answer	أختر الإجابة الصحيحة
	○ Choose two right answers	أختر الأجابتين الصحيحتين
	○ Choose all that apply	أختر الأجابات الصحيحة
True or False questions	-----	خطأ- صواب
Enter Verb	Enter Saheh	أدخل فعلاً ثلاثياً صحيحاً
	Enter Mo3tal	أدخل فعلاً ثلاثياً معتلاً
	Enter Salem	أدخل فعلاً ثلاثياً سالماً
	Enter Mahmooz	أدخل فعلاً ثلاثياً مهموزاً
	Enter Moda3af	أدخل فعلاً ثلاثياً مضعفاً
	Enter Mesal	أدخل فعلاً ثلاثياً مثالاً
	Enter Agwaf	أدخل فعلاً ثلاثياً أجوفاً
	Enter Naqes	أدخل فعلاً ثلاثياً ناقصاً

In our implementation of the **Template Method Pattern** we have three classes:

- Exercise: Is a super class that has two subclasses that are illustrated in figure (5-11):
  - YesNoExecrcise
  - MCQExcercise.

The Exercise class is a concrete class because we can instantiate an Object of Exercise to represent "Enter Verb" exercise, We implemented the **getTypeString method** this method automatically shows the title of the question according to object type as shown in the previous table. And as both YesNoExcercise and MCQExcercise are subclasses from exercise they inherit all fields and behaviors from Exercise including the getTypeString() method(Template Method) and every subclass (YesNoExcercise and MCQExcercise) provide its own implementation which overrides the implementation of Exercise class at runtime, we can interact with all exercises as objects from Exercise, so, we can call any method in Exercise including getTypeString(), but the

actual method that will be executed is dependent on the most specific type of the object (which is known as "late binding" in java).

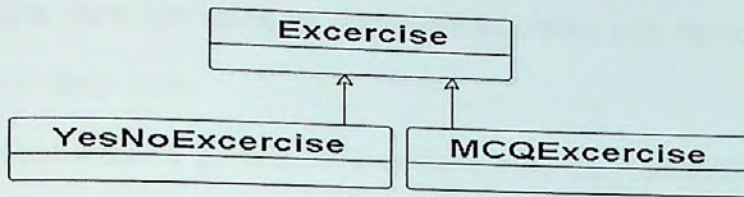


Figure (5-11): Template Method Pattern Structure in Arabic Tutor

Exercises will have a difficulty level that varies according to certain criteria; this is illustrated in table (11):

Table (11): Difficulty Level Criteria for Arabic Tutor Exercises

Question Type	Question Subtype	Difficulty level
MCQ	Choose one right answers	1
	Choose two right answers	2
	Choose all that apply	3
Enter Verb	Enter Saheh	3
	Enter Mo3tal	
	Enter Salem	
	Enter Mahmooz	
	Enter Moda3af	
	Enter Mesal	
	Enter Agwaf	
Enter Nages		
True or False		1
		2
		3

**Second Application of the Singleton Pattern in the Arabic Tutor: User Interface**

**Instantiation**

Before discussing the entry exam features we need to discuss an issue that is related to the user interface of the **Arabic Tutor**. When we implemented the **Arabic Tutor** we had a clear goal in our mind to implement a web based ITS for Teaching a subset of the Arabic language both to foreigners and to native speakers. We implemented a `UserInterfaceBundle` class that is used to

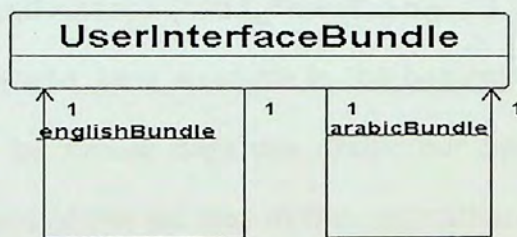
provide multilingual interface (internationalized application), in our prototype we used an Arabic/English interface however, our application is ready to represent any set of languages.

The `UserInterfaceBundle` class provides us with two bundles; one for the Arabic language and the other for the English language:

- **ArabicBundle:** Which is obtained by `getArabicBundle ()` and always gets the same Arabic bundle object.
- **EnglishBundle:** Which is obtained by `getEnglishBundle ()` and always gets the same English bundle object.

Each one of these two bundles can be represented using a **Singleton Pattern** to ensure that a class has only one instance, and provide a global point of access to it by intercepting requests to create new objects.

The Structure of the `UserInterfaceBundle` class is illustrated in figure (5-12):



**Figure (5-12): Singleton Pattern in the UserInterfaceBundle Class in the Arabic Tutor**

In order to achieve our goal and reach internationalization we used key value pair instead of hard coding the Arabic or English text in our application. The contents of the file itself are loaded in one instance of `UserInterfaceBundle`. If the file is the Arabic file then the `ArabicBundle` is loaded. However, if the file is the English file the `EnglishBundle` is loaded.

**This is an example from the key value pair file for the Arabic Language**

First\_Name\* = الاسم الأول

Last\_Name\* = اللقب

Password\* = كلمة السر

Email = البريد الإلكتروني

The ASCII of the English language is the same as the Unicode, However, this is not the same

case in the Arabic language .So we used a tool to get the Unicode for the Arabic characters

```
c:\>native2ascii -encoding utf-8 source.properties destination.properties
```

Where utf-8 is the key the code page that supports the Arabic language which means that if we wanted to have another language translation all we have to do is to change the code page. After using this command we will have a file with the destination.properties that has the Unicode as shown below:

```
First_Name=\u0627\u0644\u0623\u0633\u0645 \u0627\u0644\u0623\u0648\u0644
```

```
Last_Name=\u0627\u0644\u0644\u0642\u0628
```

```
Password=\u0643\u0644\u0645\u0629 \u0627\u0644\u0633\u0631
```

```
Email=\u0627\u0644\u0628\u0631\u064a\u062f\u0627\u0623\u0644\u0643\u062a\u0631\u0648\u0646\u0649
```

In any file in which we needed to write any text we didn't hardcode the text into the file instead we wrote <title><%=bundle.getMessage ("Add\_New\_Example") %>< /title>

Get the message that its key is add\_new\_example in the beginning of each jsp file the language is checked if it was found to be Arabic then the Arabic bundle will be used else the English bundle will be used in all the rest of the jsp files in the application.

```
<%UserInterfaceBundle bundle=UserInterfaceBundle.getArabicInstance ();%>
```

If the English language is used then we will have

```
<%UserInterfaceBundle bundle=UserInterfaceBundle.getEnglishInstance ();%>
```

The **ArabicAlphabeticBundle** class deals with the details of the Arabic language that has shapes and letters that we need to use inside our expert system so instead of using the Unicode of these letters and shapes inside our application leading to non readable code we used an English name for every Unicode so as to improve the readability of the expert module.

As an example, some sample rules from our expert system are listed below to show the utilization of the English names for the Unicode of every Arabic letter.

```
(defrule is-saheh
```

```
  (test (not (call ?*infinitive* hasLetter "alef")))
```

```
(test (not (call ?*infinitive* hasLetter "waw")))
```

```
(test (not (call ?*infinitive* hasLetter "yeh")))
```

=>

```
(assert (verb-type saheh))
```

```
(assert (verb-type salem))
```

### Second Application of the New Flyweight Pattern in the Arabic Tutor: Arabic Alphabetic Letters and Shapes Instantiation

Since all the students who will use the **Arabic Tutor** need to use the Arabic Alphabet we did not allow students to instantiate this shared object in order to avoid duplication. The application is initialized by loading all the needed Arabic alphabetic letters and shapes (fatha, damma, kasra, shadda, sukun) using the **New Flyweight Pattern** to instantiate Arabic Alphabetic Letters and Shapes since it describe how to share objects to allow their use at fine granularities without prohibitive cost.

We did not to associate shapes with letters to protect both the letter and the shape intrinsic state and thus we can reach the following situation:

- Use the letter by itself.
- Use the letter with one shape (primary shape).
- Use the letter with two shapes (primary shape and secondary shape).

All these forms of the letter are considered as one character that is needed by the expert system in order to apply the grammar rules upon it. Students use the already instantiated objects of letters and shapes. The **New Flyweight Pattern** structure is illustrated in figure (5-13):

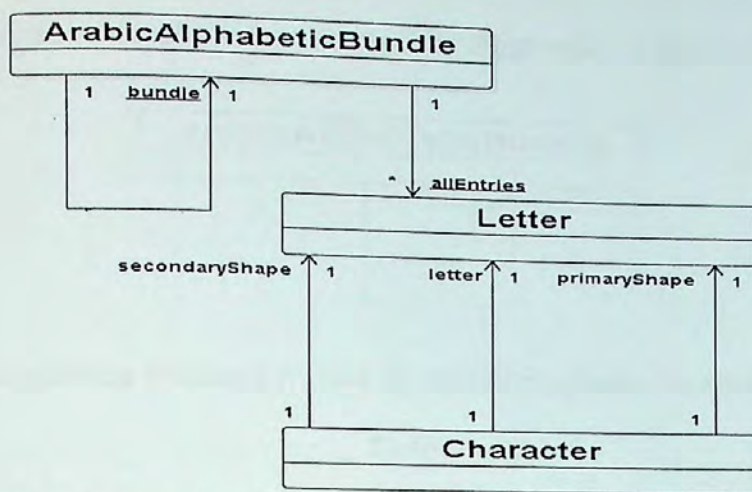


Figure (5-13): New Flyweight Pattern Structure in the ArabicAlphabeticBundle Class in the Arabic Tutor

**New Flyweight Participants in the Arabic Tutor:**

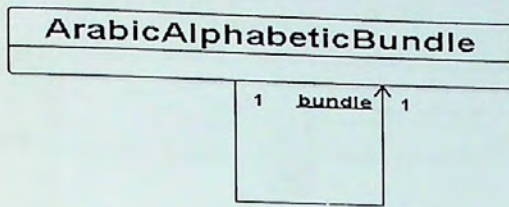
1. **ConcreteFlyweight** is a sharable object. Any state it stores must be intrinsic; that is, it must be independent of the ConcreteFlyweight object's context. This participant can be represented by an instance of the Letter class.
2. **Unshared ConcreteFlyweight:** It's common for UnsharedConcreteFlyweight objects to have ConcreteFlyweight objects as children at some level in the Flyweight object structure. This participant can be represented by an instance of the Character class because it depends on the context of using the letter. A character can be represented as a letter with no shape or a letter with shape or a letter with two shapes.
3. **Flyweight Factory:** Is responsible for creating and managing Flyweight objects and Ensuring that Flyweights are shared properly. When a client requests a Flyweight, the FlyweightFactory object supplies an existing instance or creates one, if none exists. This participant can be represented by the instance of the ArabicAlphabeticBundle class.
4. **Client:** Is responsible for maintaining a reference to Flyweight(s) and storing the extrinsic state of Flyweight(s). This participant can be represented by the expert module.

**Third Application of the Singleton Pattern in the Arabic Tutor: Arabic Alphabetic Letters Instantiation**

Since all students use the same Arabic Alphabet thus we need to implement it using **Singleton Pattern** to ensure the existence of a single instance.



The Structure of the ArabicAlphabeticBundle class is illustrated in figure (5-14):



**Figure (5-14): Singleton Pattern in the ArabicAlphabeticBundle Class in the Arabic Tutor**

The entry exam is composed of 9 questions per topic that consists of 3 questions per difficulty level that are chosen randomly by the **Exam Generator Component** of the Tutor Module. This is illustrated in figure (5-15):

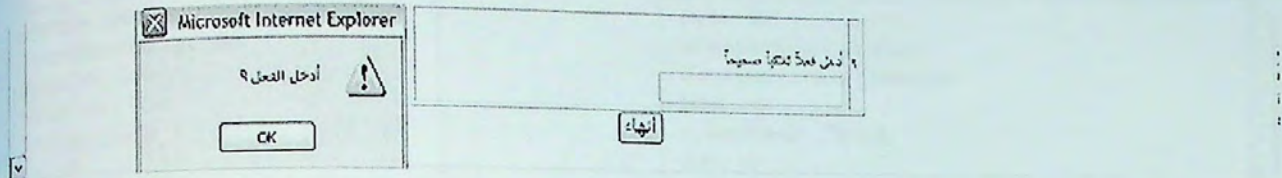
	<p>أُستخرج التعيين تصحيحين</p> <p>لعب <input type="checkbox"/></p> <p>باع <input type="checkbox"/></p> <p>أفذا <input type="checkbox"/></p> <p>لقي <input type="checkbox"/></p>	
	<p>أُستخرج التعيين تصحيحين</p> <p>رعى <input type="checkbox"/></p> <p>دلى <input type="checkbox"/></p> <p>دعا <input type="checkbox"/></p> <p>أمن <input type="checkbox"/></p>	
	<p>تبدل الاسم هو أحد الأسماء التي تمثل</p> <p>صواب خطأ</p> <p><input type="radio"/> <input type="radio"/></p>	
	<p>أُستخرج الأفعال تصحيحاً</p> <p>سأل <input type="checkbox"/></p> <p>مأ <input type="checkbox"/></p> <p>ضم <input type="checkbox"/></p> <p>لعب <input type="checkbox"/></p>	
	<p>أُستخرج لعدلاً تصحيحاً</p>	

**Figure (5-15): Entry Exam Sample Exercises**

**Third Application of the Shield Pattern in the Arabic Tutor: Preventing Students from not answering a Question**

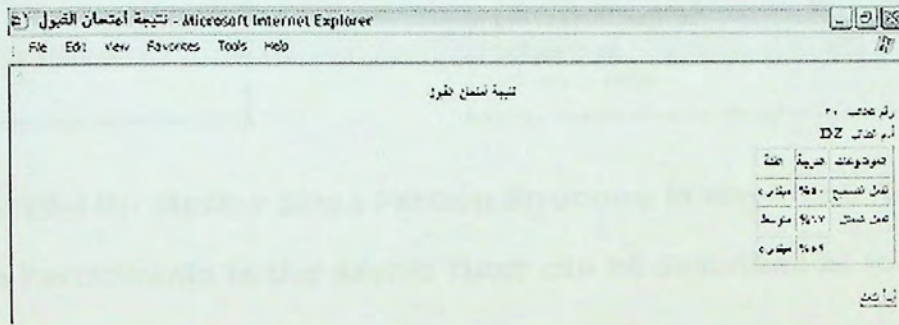
The student is disabled from finishing an exam unless he answers all the exam questions and

this is considered another utilization of the **Shield Pattern**.



**Figure (5-16): Shield Pattern in the Arabic Tutor –Protecting Student from Unanswering a Question**

After the student finishes the entry exam a score sheet will be shown that illustrates his current knowledge level per topic. The entry exam score sheet is illustrated in figure (5-17):



**Figure (5-17): Entry Exam Score Sheet (\*)**

Names are abbreviated to protect student's privacy

#### 5.4.12 Master Slave Pattern in the Arabic Tutor

The **Master Slave Pattern** is used to distribute work between slave components so as to allow for computational accuracy and uses the results provided by each partial process to compute the result of the whole operation. The **Master Slave Pattern** is used to compute the exam results of each student according to their grade in each particular exercise within the exam.

The structure of the **Master Slave Pattern** is illustrated in figure (5-18):

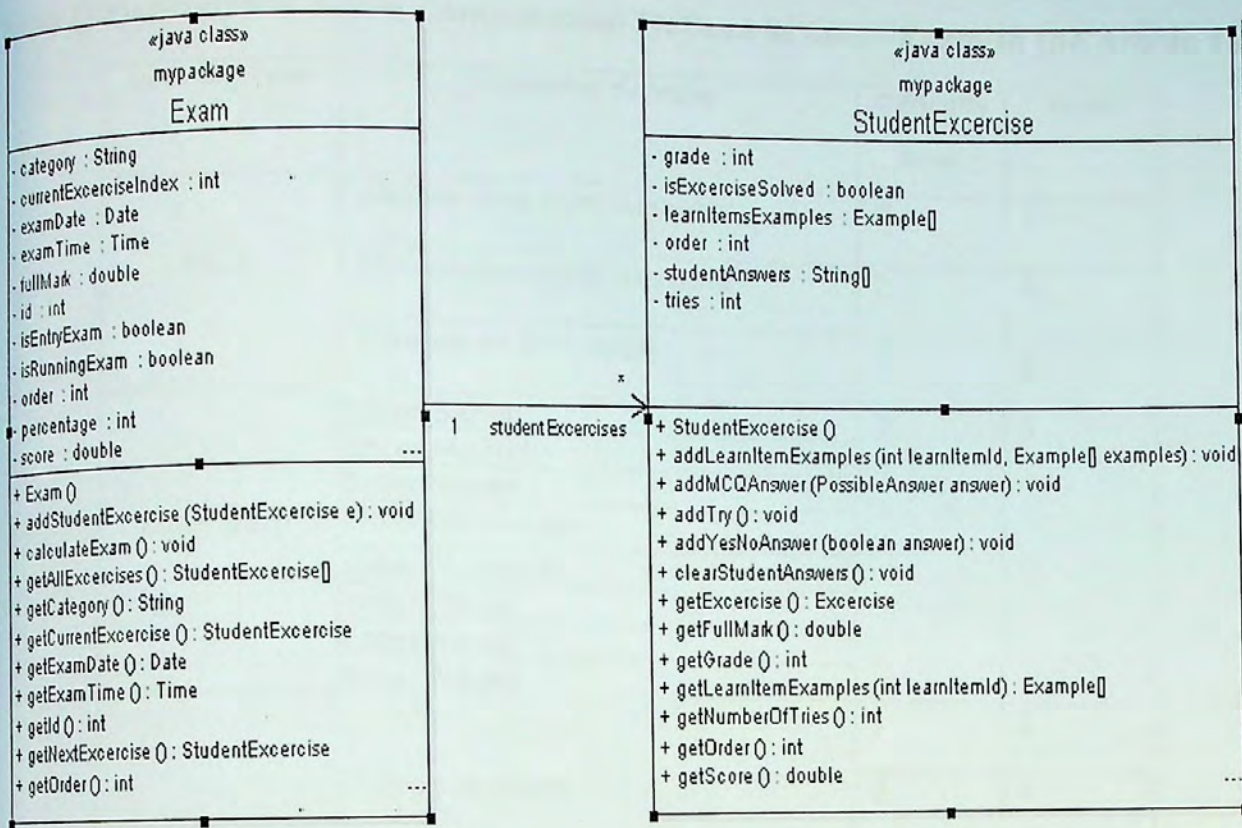


Figure (5-18): Master Slave Pattern Structure in the Arabic Tutor

The Master Slave Participants in the Arabic Tutor can be described as follows:

- **Master:** That is represented by the Exam object. Each exam contains a collection of StudentExercise. **This class is responsible for the following functionality:**
  - Displaying exercises in the right sequence by utilizing the **New Strategy Pattern.**
  - Calculating the overall student score based on scores calculated from StudentExercise objects.
- **Slave:** That is represented by a collection of StudentExercise.

All students will be given a score on answering questions that will vary according to the question difficulty level. This score computation method in the **Arabic Tutor** can be explained in the following table:

Table (12): Students' Score Computation Method in Entry Exam in the Arabic Tutor

Question Type	Question Subtype	Difficulty level	Score
MCQ	Choose one right answers	1	2
	Choose two right answers	2	4
	Choose all that apply	3	6
Enter Verb	Enter Saheh	3	6
	Enter Mo3tal		
	Enter Salem		
	Enter Mahmooz		
	Enter Moda3af		
	Enter Mesal		
	Enter Agwaf		
	Enter Naqes		
True or False		1	2
		2	4
		3	6

The system uses this entry exam to initialize the student model. This entry exam helps to make a preliminary categorization for student levels per topic and per learning item; this categorization is then refined by observing the student while working with the system.

#### 5.4.13 New Adapter Pattern in the Arabic Tutor

##### First Application of the New Adapter Pattern in the Arabic Tutor: Representing Student

##### Status per Exercise

The **New Adapter Pattern** is used to convert the interface of a class into another interface clients expect and lets classes work together that couldn't otherwise because of incompatible interfaces [7]. The **New Adapter Pattern** was used with the implementation of the StudentExcercise Class that acted as an Adapter that is capable of converting the interface of the Exercise class into another interface that the client expects. And to provide us with the functionality that the adapted class (Exercise) doesn't provide.

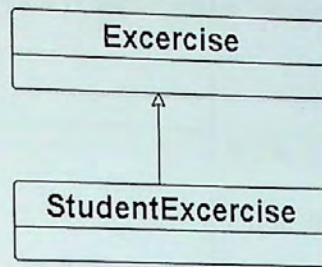


Figure (5-19): New Adapter Pattern Structure in the Arabic Tutor (Excercise-StudentExcercise)

Each StudentExcercise is concerned with reflecting the student specific status per exercise. **This class tracks the following information:**

- What is the exercise type?
- What are the possible answers to the exercise?
- What are the right answers to the exercise?
- Did the student view this exercise or not? Students can not be given the same exercise twice; as soon as a certain exercise is represented to a student it is removed from the pool of available exercises for this student.
- The student answer on this exercise and whether he answered correctly or not, this will help to understand any misconception that the student might have.
- The student score per exercise.

### Third element in User Goals Pattern: Stereotype

Stereotype applies to the student, which is the group of learners he belongs to based on some predefined presuppositions in terms of knowledge level, learning and cognitive styles.

After the student takes this entry exam, he will view his score sheet that indicates his current stereotype per topic. This stereotype is used to initialize the student model. Students will be categorized into 4 stereotypes according to their score in the entry exam. These stereotypes reflect their knowledge level per topic. These stereotypes are illustrated table (13):

**Table (13): Students' Stereotypes**

Category	Score per topic
Expert	90%-100%
Advanced	80%-90%
Intermediate	65%-80%
Beginners	Less than 65%

It is important to notice that this categorizing of students will be determined per topic; i.e., a student can be an expert in a certain topic and a beginner in another topic.

**Fourth element in User Goals Pattern: Usage data**

Usage Data includes the following information:

- **Topics:** Whether they are already covered, need to be covered or needn't to be covered.
- **Current Student Level Per Topic:** Whether a beginner, intermediate, advanced or expert level.
- **Viewed Teaching Material:** Including topics learn items, examples or exercises.

**5.4.14 New Student Maintenance Pattern in the Arabic Tutor**

Usage data can be acquired through both the **New Student Model Maintenance Pattern** and the **New Adapter Pattern**. During the course of interaction current student knowledge level develops and builds up. Since the adaptation is to a large extent based on student knowledge and usage data, changes should definitely be recorded and be related to a "cause / result" [28].

Thus, the student model must be maintained to reflect the new realities. The method that was utilized by the **New Student Model Maintenance Pattern** was to update the student model description in the **Arabic Tutor** using the results of the exam generator component that measures current student knowledge level per topic.

## Second Application of New Adapter Pattern in the Arabic Tutor: Representing Student Status per Topic

The Topic class doesn't match the domain specific interface that the application requires because it doesn't reflect the student specific status per topic. Thus the Topic class does not provide us with usage data. This class doesn't represent the following information:

- The student stereotype per topic whether he is a Beginner, Intermediate, Advanced or an Expert student.
- What learning item is currently being learned inside the topic by the student?
- How many times the student viewed a certain topic?
- What are the exams that he took on a certain topic?
- Student grades on every exam concerning this topic.

The **New Adapter Pattern** was used to provide us with this usage data and to convert the interface of a class into another interface clients expect and let classes work together that couldn't otherwise because of incompatible interfaces [7].

The **New Adapter Pattern** converts the interface of a class (Topic) into another interface that the client expects (StudentTopic) and provides us with functionality that the adapted class (Topic) doesn't provide. This is illustrated in figure (5-20):

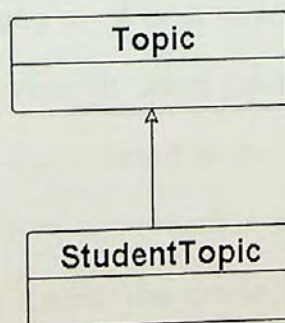


Figure (5-20): New Adapter Pattern Structure in the Arabic Tutor (Topic-StudentTopic)

### Short Term Goal Handling in the Arabic Tutor:

Short term goals are goals that are valid for a shorter period of time, for example, if the long term goal of a certain student was "to learn topic X", then the short term goal is the presentation of the appropriate learn items and examples that he needs to study or view in

order to gain the required knowledge to master the chosen topic.

This is accomplished with the help of the Tutor module component of the **Arabic Tutor**. The short term goal handling in the **Arabic Tutor** begins after the student signs up and determines his long term goals from using the system. The results of the entry exam per topic will be used to control the teaching material that will be viewed by the student .This is illustrated table (14):

**Table (14): Teaching Material According to Student Level.**

Student Stereotype	Topic Explanatory Text	Learning Item Explanatory Text	Number of Learn Item Examples
Beginner	Yes	Yes	6
Intermediate	No	Yes	4
Advanced	No	Yes	2
Expert	No	No	1

In this phase, the student needs to perform a complex task consisting of several subtasks where decisions that need to be made in each subtask may not be known to the student. This complex task is represented by the student long term goal which is mastering a certain topic. This long term goal can be divided into a number of short term goals. Thus the complex task can be divided into a number of a subtasks represented in the learn items that are included inside this topic and that needs to be studied in a particular order to preserve the predecessor and successor relationship between learn items. The predecessor-successor relationship may not be known to the student and it is entered initially by the teacher with the help of the **Early Bird Pattern** and the **Course Creation and Customization Pattern**.



#### 5.4.15 New Strategy Pattern in the Arabic Tutor

In the **Arabic Tutor**, we used the **New Strategy Pattern** to allow the student to reach their short term goals. The **New Strategy Pattern** defines a family of algorithms, encapsulates each one, and makes them interchangeable and lets the algorithm vary independently from clients that use it. **New Strategy Pattern** is needed when an algorithm uses data that clients shouldn't know about and helps to avoid exposing complex, algorithm specific data structures [7].

In the **Arabic Tutor**, the **New Strategy Pattern** was used in defining a sorting algorithm for the following objects:

- **Topics learn items, exercises, examples:** These objects are sorted according to sequence number. The sequence number is entered initially by the teacher using both the **Early Bird Pattern** and the **Course Creation and Customization Pattern**.
- **Student's Topics:** Sorted according to both the sequence and the student's stereotype per topic. This means that after we give the student the entry exam, according to his score in the exam we determine the list of topics that needs to be learned by him. If he needs to learn two topics that has the same sequence number then we will start with the topic that he has the lowest stereotype in (if both topics has the same sequence number yet the student level is beginner in one topic and intermediate in the other topic we usually start with the topic in which the student is a beginner).

We have three participants in the **New Strategy Pattern**:

1. **Strategy:** Represents an interface common to the supported algorithm. This participant is represented by the sortable interface.
2. **ConcreteStrategyClient:** Implements the algorithm using the Strategy interface. This Participant is represented by sortObjects () static method in the class utilities.

ConcreteStrategyClient is an object that uses the **New Strategy Pattern**. In the **Arabic Tutor**

we have the following clients:

- Topic
- LearnItem
- Exercise

- Example
- Exam
- StudentTopic

3. **Context:** In the **Arabic Tutor**, the Context is the static method "sortObjects()" that takes an array of Sortable objects (ConcreteStrategyClient) to sort them based on the strategy implemented in the ConcreteStrategyClient which decides the order of two objects and the context is responsible for performing the overall sorting of all objects.

The New Strategy Pattern Structure in the Arabic Tutor is illustrated in figure (5-21):

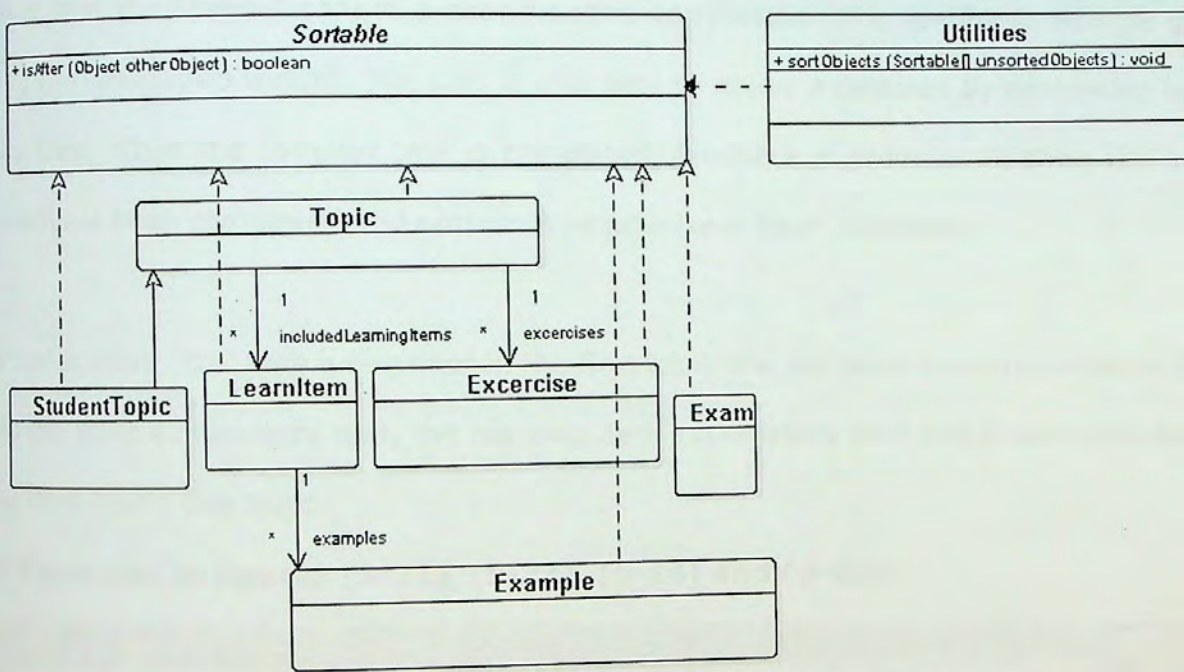


Figure (5-21): New Strategy Pattern Structure in the Arabic Tutor

In our context in the Arabic Tutor, we followed the following steps:

1. We created the Sortable interface which implements the isAfter () method).
2. We created a method called sortObjects that take an array of any objects implementing the Sortable interface so these objects are assured to implement the isAfter method which decides the relative ordering between two objects only and the method logic is responsible for sorting the whole array.

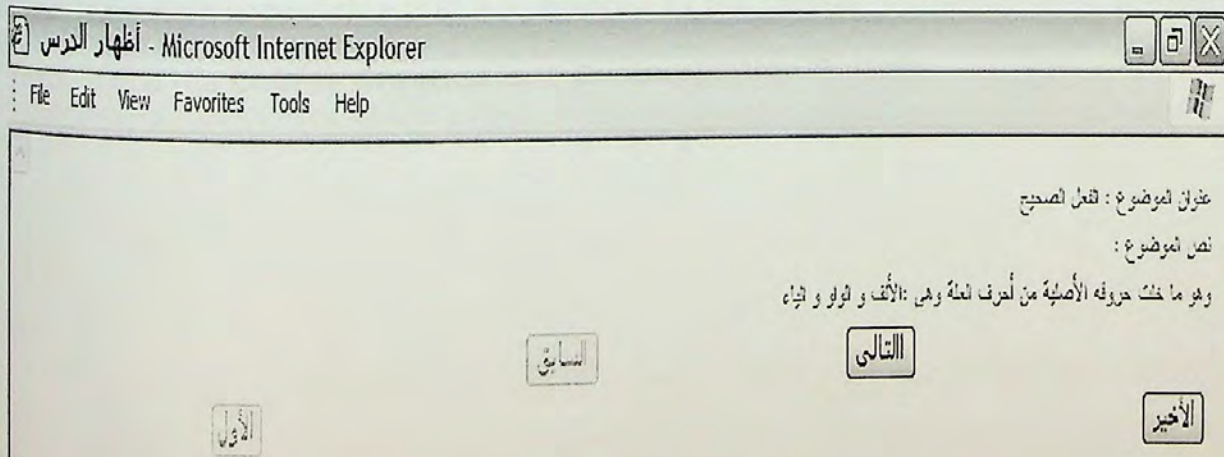
#### 5.4.16 Wizard Pattern in the Arabic Tutor

In addition, we used the **Wizard Pattern** to allow the student to reach their short term goals by taking them through the entire complex task one step at a time. This was achieved by using a navigation widget represented by a button. The navigation buttons suggest to the students that they are navigating a path with steps. Each task is presented in a consistent fashion enforcing the idea that several steps are taken.

If the student cannot start the next task before completing the current one, feedback is provided indicating that the student cannot proceed before completion; this feedback may be given by disabling the navigation widget. The user is also able to revise a decision by navigating back to a previous task. When the complex task is completed, feedback is provided to show the user that the tasks have been completed and optionally results have been processed.

And as our student "DZ" was a beginner in the first topic she will view teaching material that includes the topic explanatory text, the learning item explanatory text and 6 examples per learning item inside this topic.

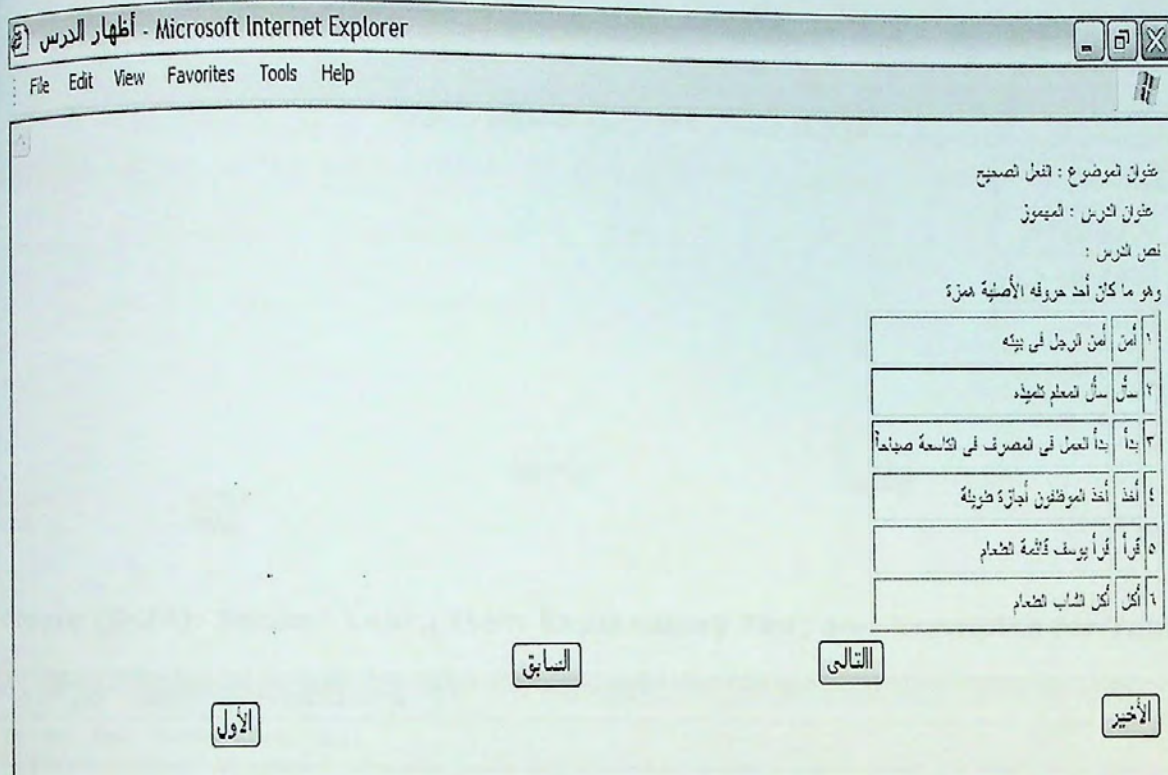
This is illustrated in figures (5-22), (5-23), (5-24) and (5-25):



**Figure (5-22):Topic1 Explanatory Text**

Our student "DZ" was interested in two topics, however, when he/she chooses to begin learning the system will utilize the **New Strategy Pattern** and show her the predecessor topic first. In

In addition we also utilized the **Wizard Pattern** using a navigation widget to guide the student to the next learning task.



**Figure (5-23): First Learn Item Explanatory Text and examples for Topic1**

Figure (5-23) shows how the **Arabic Tutor** leads the student in his short term goals achievement by showing the learn items that belong to this topic one at a time while preserving the correct learn item sequencing that was entered by the teacher with the help of the **Early Bird Pattern** and the **Course Creation and Customization Pattern**.

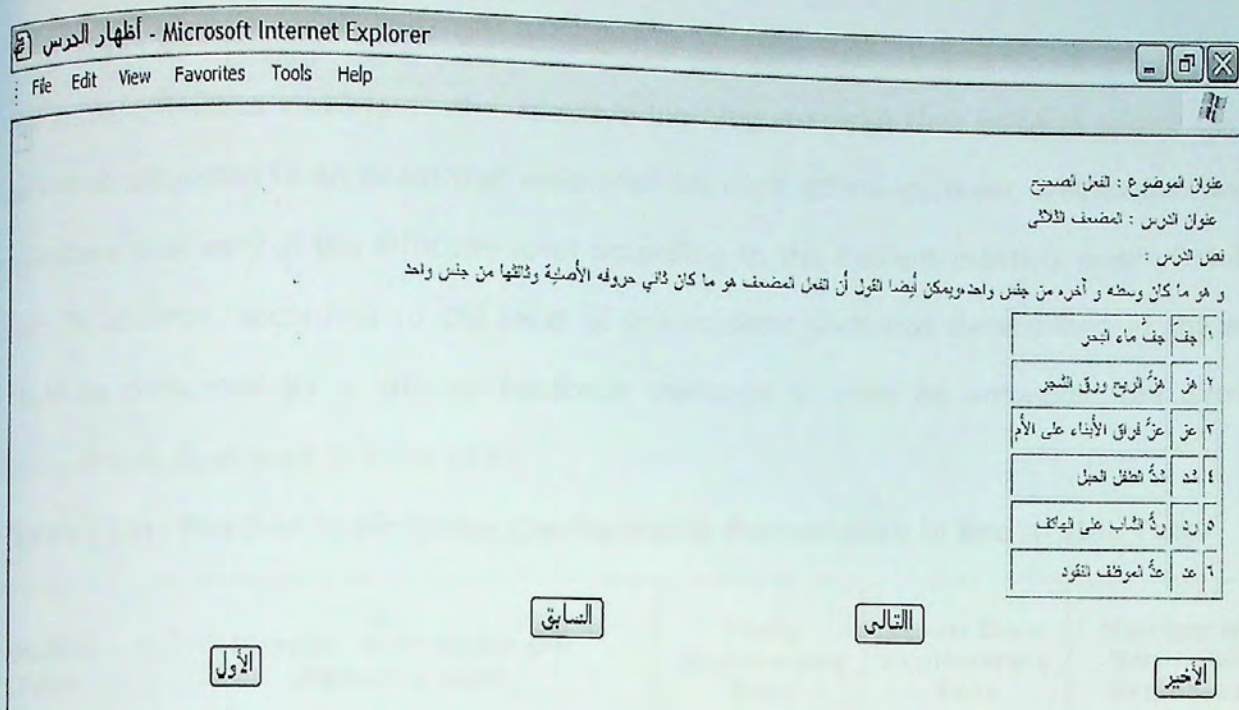


Figure (5-24): Second Learn Item Explanatory Text and Examples for Topic1

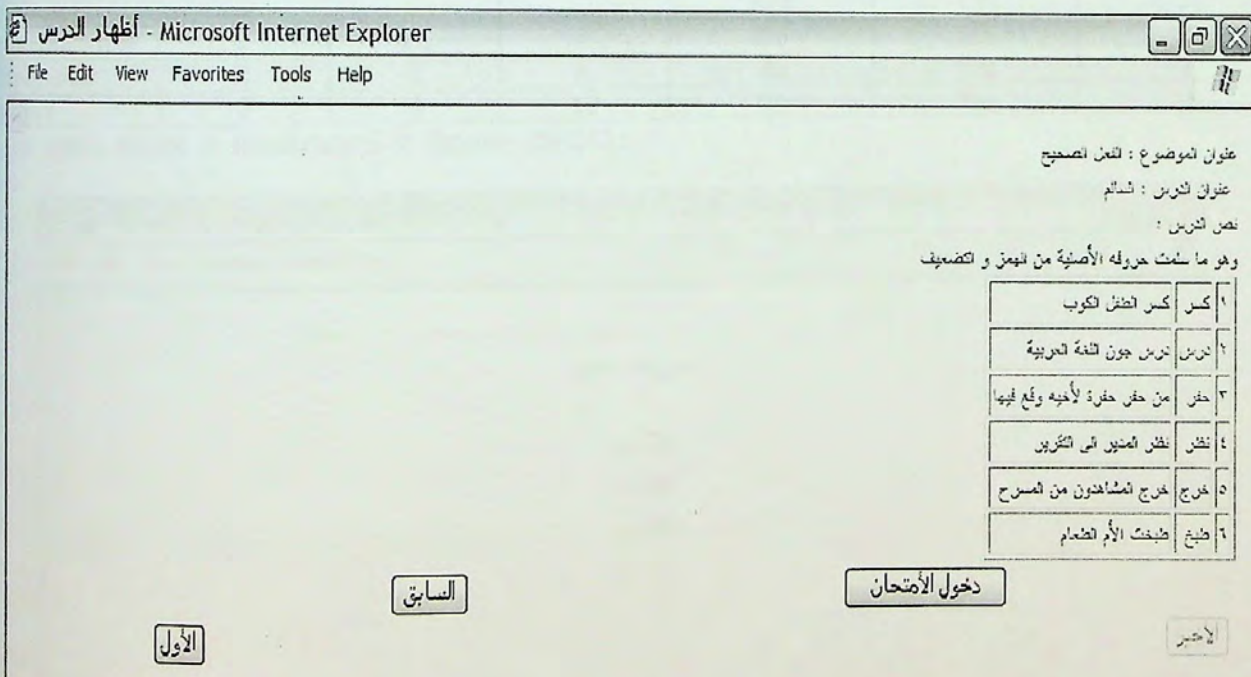


Figure (5-25): Third Learn Item Explanatory Text and Examples for Topic1

As it can be noticed from figures (5-23), (5-24) and (5-25) the **Arabic Tutor** arranges the learn items and examples that will be viewed by the student in the suitable number and quantity that suits the current knowledge level of the student, thus relieving the student from the burden of determining how to accomplish short term goals.

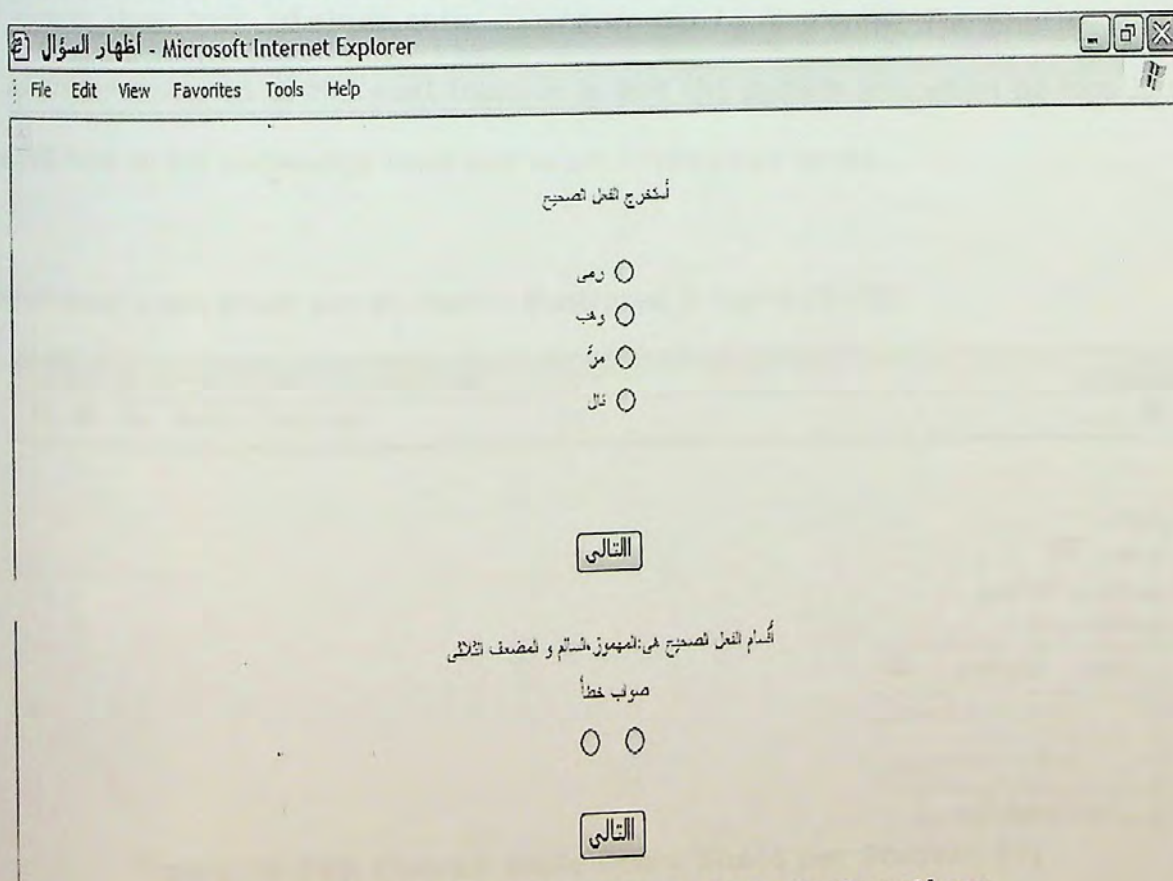
• **Per Topic Exam:**

After the student finishes viewing all the adaptive learning material that belongs to the current topic he will be subjected to an exam that measures his topic efficiency level. This exam consists of 12 questions that vary in the difficulty level according to the current mastery level of student per topic. In addition, according to the level of the student that was determined in the entry exam he'll be presented by a tailored feedback message in case he answered the question incorrectly. This is illustrated in table (15):

**Table (15): The Per Topic Exam Configurable Parameters in the Arabic Tutor**

Student Level	Number of Exercise per difficulty level			Topic Explanatory Text	Learn Item Explanatory Text	Number of Remedial examples
	Difficulty Level 1	Difficulty Level 2	Difficulty Level 3			
Beginner	12	0	0	Yes	Yes	3
Intermediate	0	12	0	No	Yes	2
Advanced	0	6	6	No	Yes	1
Expert	0	0	12	No	No	0

The per topic exam is illustrated in figure (5-26):



**Figure (5-26): Per Topic Exam Sample Questions**

The **Arabic Tutor** prevents the student from making an action that has irreversible side effect by disabling the action that will lead him to this situation.

This is illustrated in figure (5-27) that utilizes the **Shield Pattern** by disabling the student from finishing an exam unless he fully answered all the exam questions including those questions which have more than one correct answer.

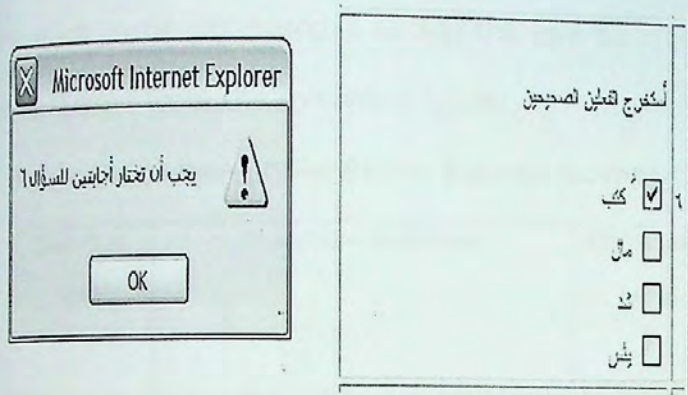


Figure (5-27): **Shield Pattern in the Arabic Tutor – Protecting the Student from Semi Answering a Question**

After taking this exam the students are presented by a score sheet that identifies their score in all the exams they took whether entry exams or per topic exams. The student is given the option to either move on to the next topic or to exit the system and when he logs in the next time he will find all his knowledge level and exam information saved.

The overall topic score sheet per student is illustrated in figure (5-28):

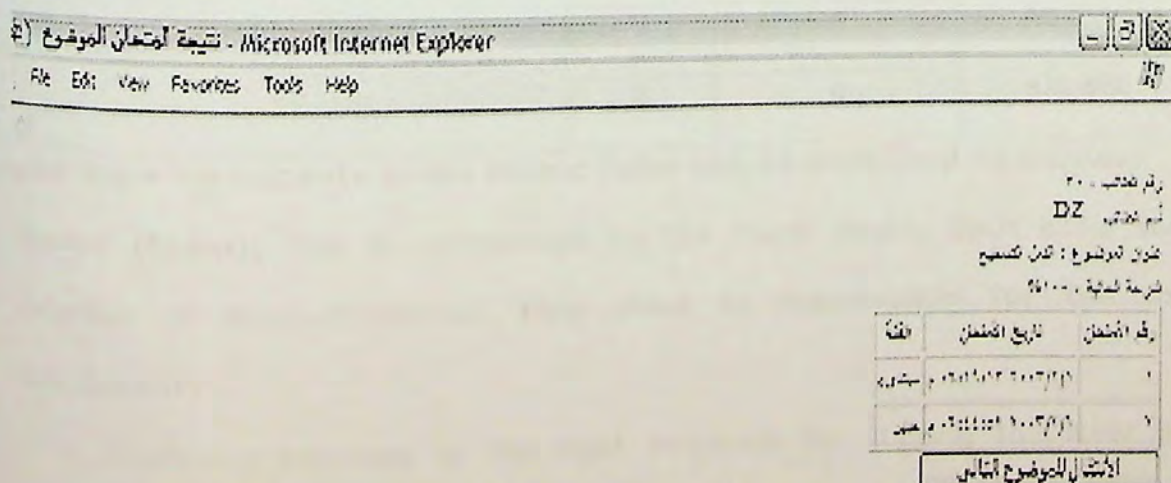


Figure (5-28): **Overall Topic Score Sheet per Student (\*)**

\* Names are abbreviated to protect student's privacy

It is important to notice in figure (5-28) that the **New Strategy Pattern** is used to implement this exam score sheet so as to sort all the exams taken by the student in chronological order.

This per topic exam score sheet is implemented with the help of the **Master Slave Pattern**. **Master Slave Pattern** is used to compute the exam results of each student according to their grade in each particular exercise within the exam.

This is illustrated from the following table:

**Table (16): Per Topic Exam Scores Computation Method in the Arabic Tutor**

Question Type	Question Subtype	Difficulty level	Score if correct in the first attempt	Score if correct from the second attempt
MCQ	Choose one right answers	1	2	1
	Choose two right answers	2	4	2
	Choose all that apply	3	6	3
Enter Verb	Enter Saheh	3	6	No Second Attempt allowed
	Enter Mo3tal			
	Enter Salem			
	Enter Mahmooz			
	Enter Moda3af			
	Enter Mesal			
	Enter Agwaf			
Enter Naqes				
True or False		1	2	No Second Attempt allowed
		2	4	
		3	6	

The **Master Slave Participants in the Arabic Tutor** can be described as follows:

- **Master (Exam):** That is represented by the Exam object. Each exam contains a collection of StudentExcercise. **This class is responsible for the following functionality:**
  - Displaying exercises in the right sequence by utilizing the **New Strategy Pattern**.
  - Calculating the overall student score based on scores calculated from



StudentExcercise objects.

- **Slave (StudentExcercise):** That is represented by a collection of StudentExcercise. Each StudentExcercise is concerned with reflecting the student specific status per exercise. **This class utilizes the New Adapter Pattern to track the following information:**

- What is the exercise type?
- What are the possible answers to the exercise?
- What are the right answers to the exercise?
- Did the student view this exercise or not? Students can not be given the same exercise twice; as soon as a certain exercise is represented to a student it is removed from the pool of available exercises for this student.
- The student answer on this exercise and whether he answered correctly or not, this will help to understand any misconception that the student might have.
- The student score per exercise.
- The number of student tries for solving this exercise.
- Did the student use out all his tries in this exercise or not yet?
- The examples that is related to this exercise, this means that if the student answers incorrectly he will be shown some teaching material to revise the learning item(s) that is covered by this exercise, this teaching material includes some examples so we should keep a record with the examples that is related to this exercise.

If the student chooses to move to the next topic he will be presented by adaptive learning material that is suitable to his current knowledge level as mentioned earlier. This can be shown in figure (5-29).

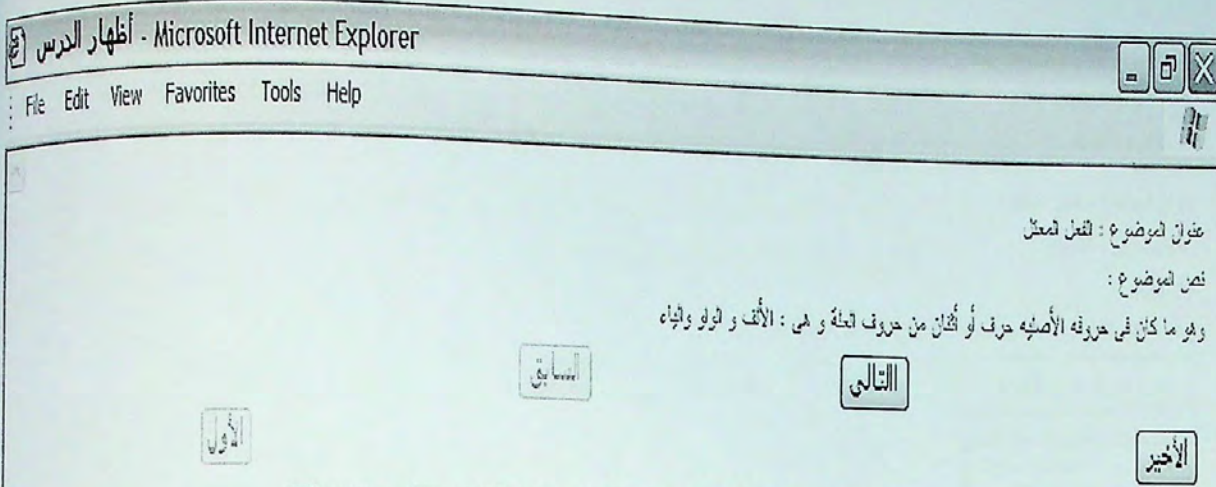


Figure (5-29): Topic2 Explanation Text

Then the student will be presented by the learning item text and examples that vary in number according to the current student level, as the student current level in this topic is intermediate level he will view 4 examples per learning item as illustrated in figures (5-30), (5-31) and (5-32).

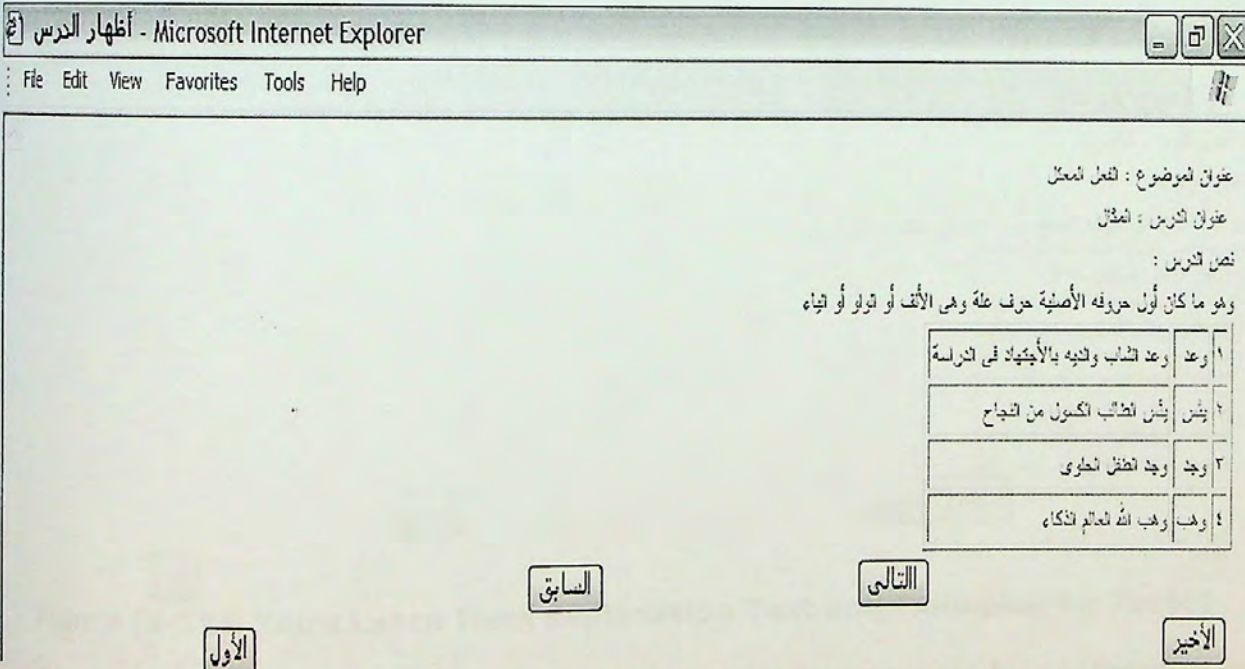


Figure (5-30): First Learn Item Explanation Text and Examples for Topic2

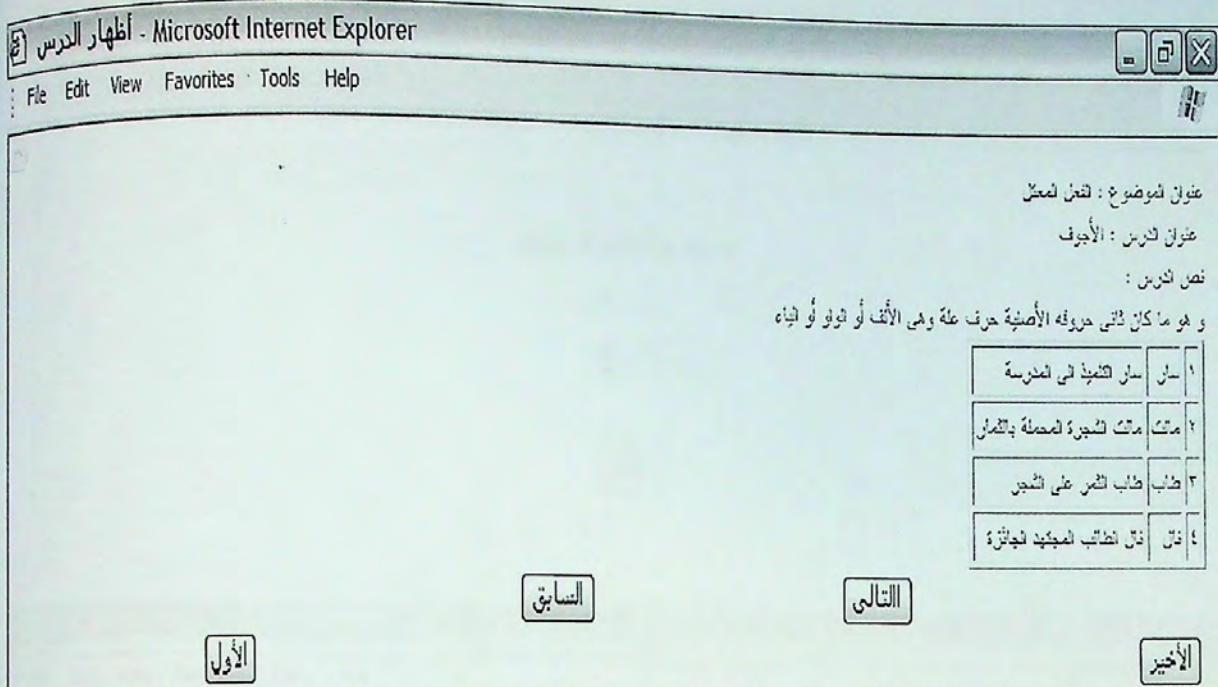


Figure (5-31): Second Learn Item Explanation Text and Examples for Topic2

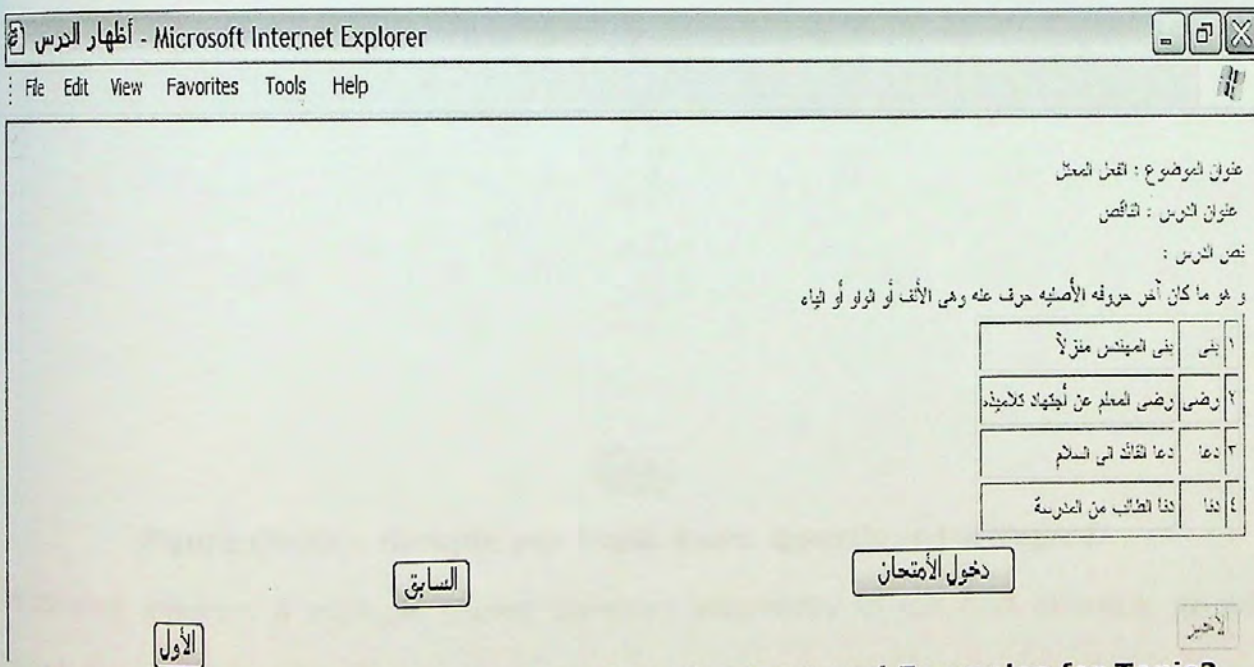
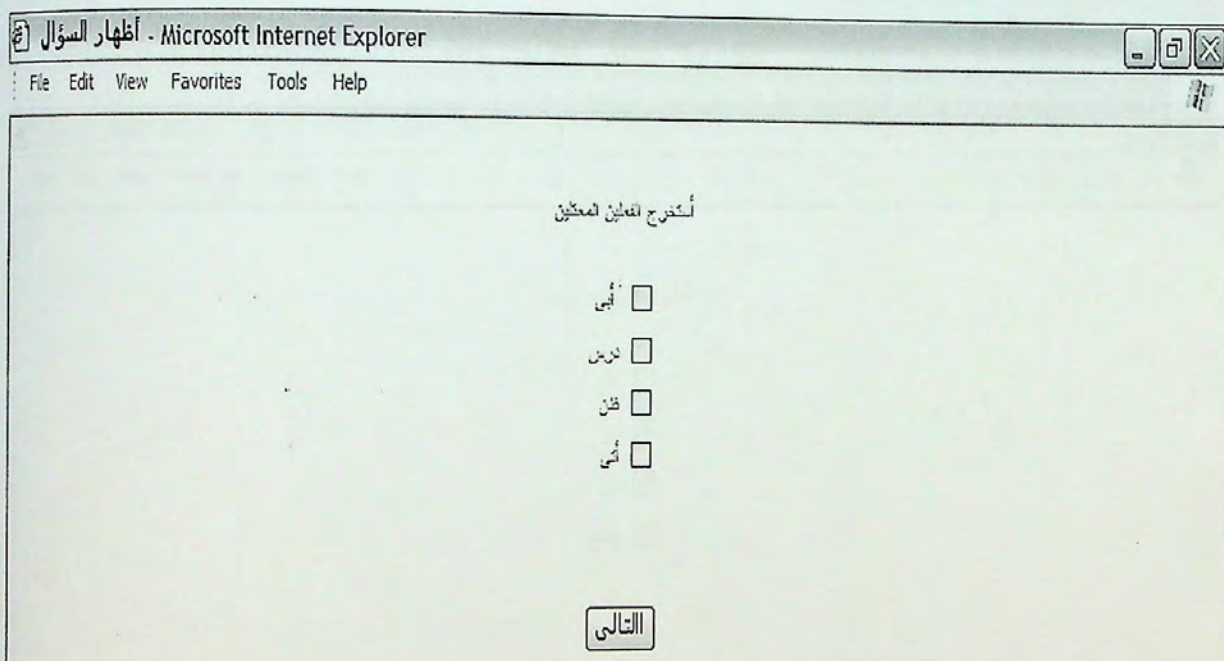
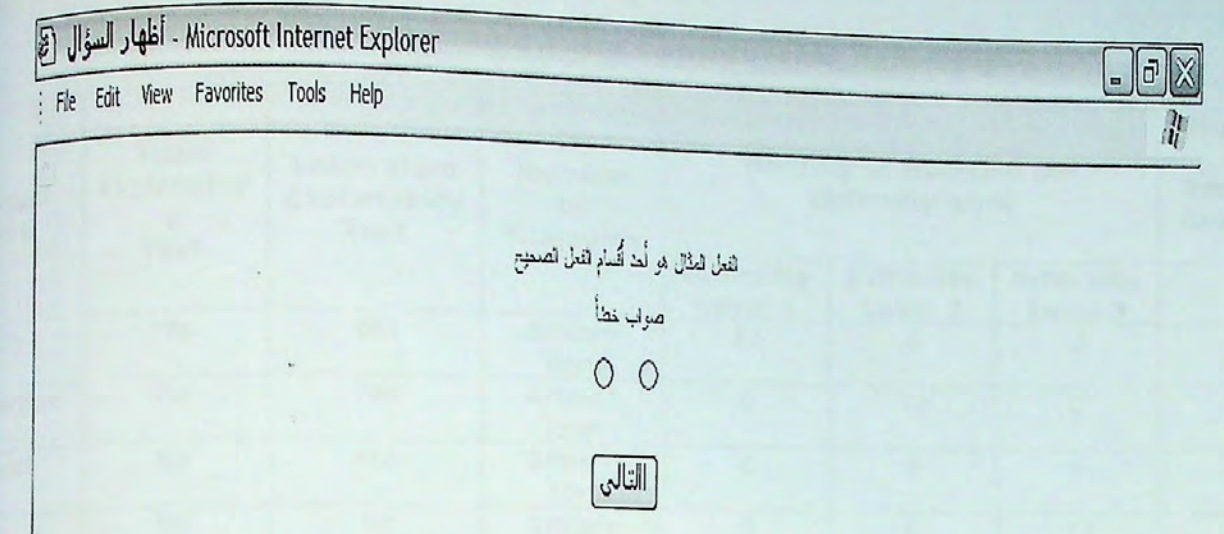


Figure (5-32): Third Learn Item Explanation Text and Examples for Topic2

Afterwards, if the student decided to enter the exam he will be presented by a tailored exam that varies in the difficulty level per exercise according to the current student level. A sample from this exam questions is illustrated in figure (5-33):



**Figure (5-33): Sample per Topic Exam Questions for Topic2**

If the student answers a multiple choice question incorrectly in the first attempt, he will be presented by a remedial textual explanation that acts as tailored feedback which varies according to his current knowledge level as illustrated in the following table.

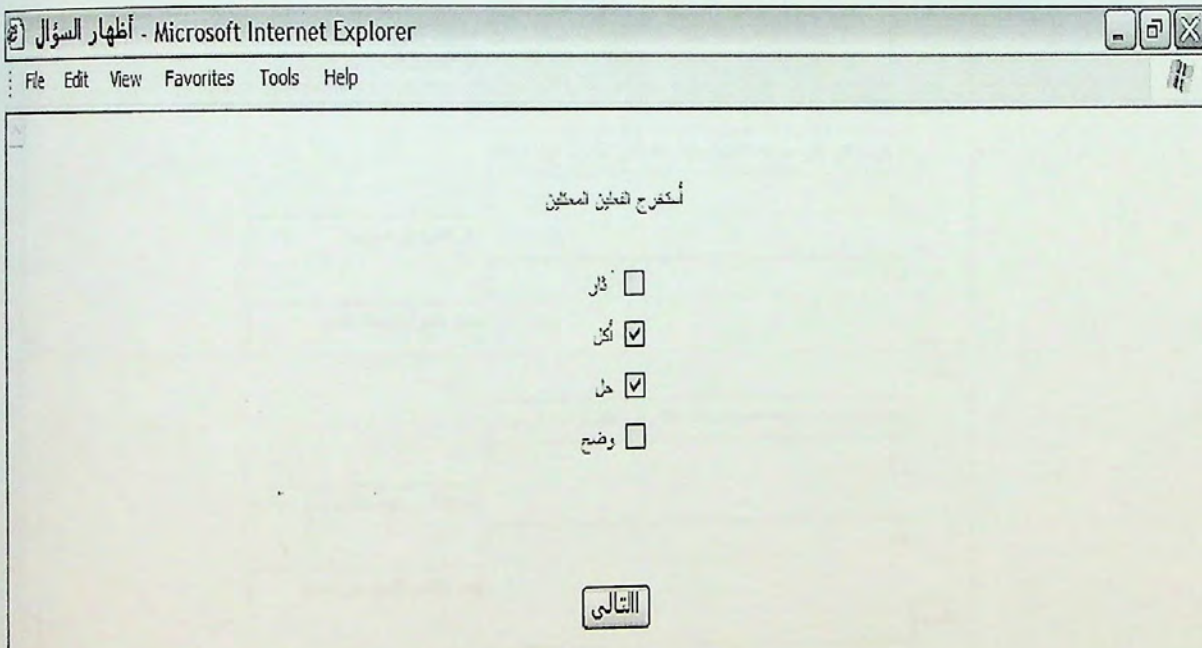
**Arabic Tutor Explanation Generator Component:**

We provided different textual explanations to students that depended on configurable parameters that can be listed as follows:

**Table (17): The Arabic Tutor Configurable Parameters**

Student Level	Topic Explanatory Text	Learn Item Explanatory Text	Number of examples	Number of Exercise per difficulty level			Number of Remedial examples
				Difficulty Level 1	Difficulty Level 2	Difficulty Level 3	
Beginner	Yes	Yes	6/learn item	12	0	0	3
Intermediate	No	Yes	4/learn item	0	12	0	2
Advanced	No	Yes	2/learn item	0	6	6	1
Expert	No	No	1/learn item	0	0	12	0

Tailored feedback is illustrated in figures (5-34) and (5-35):



**Figure (5-34): Wrong Student Answer on an MCQ Question**

Since the student answer is a wrong answer he will view the following screen:

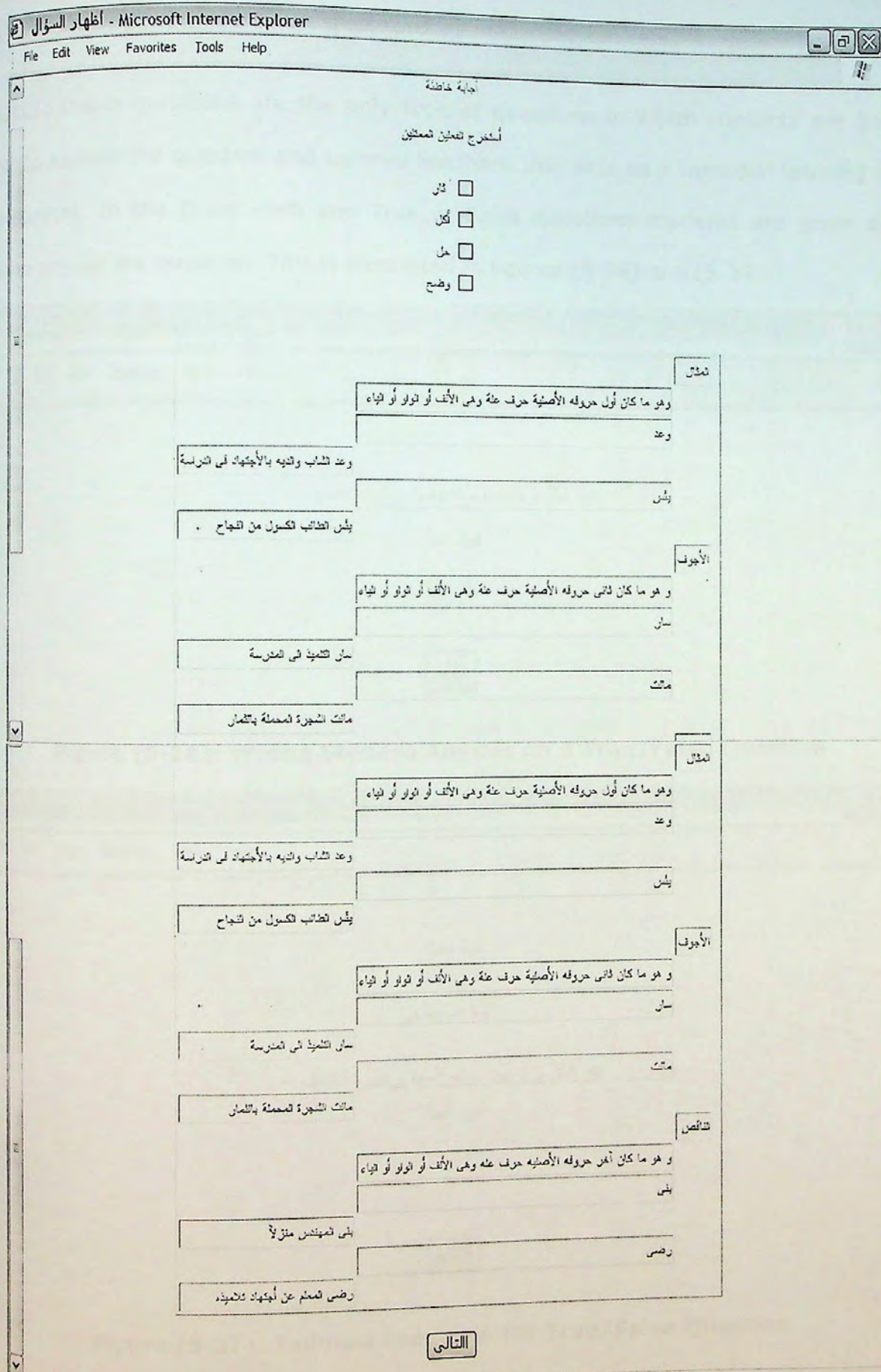


Figure (5-35): Tailored Feedback

Then the student will be given a second chance to solve the question, if answered correctly he will take half the original grade of the question.

The multiple choice questions are the only type of questions in which students are given two attempts to answer the question and tailored feedback that acts as a remedial learning material for the student. In the Enter Verb and True or False questions students are given only one attempt to answer the question. This is illustrated in figures (5-36) and (5-37):

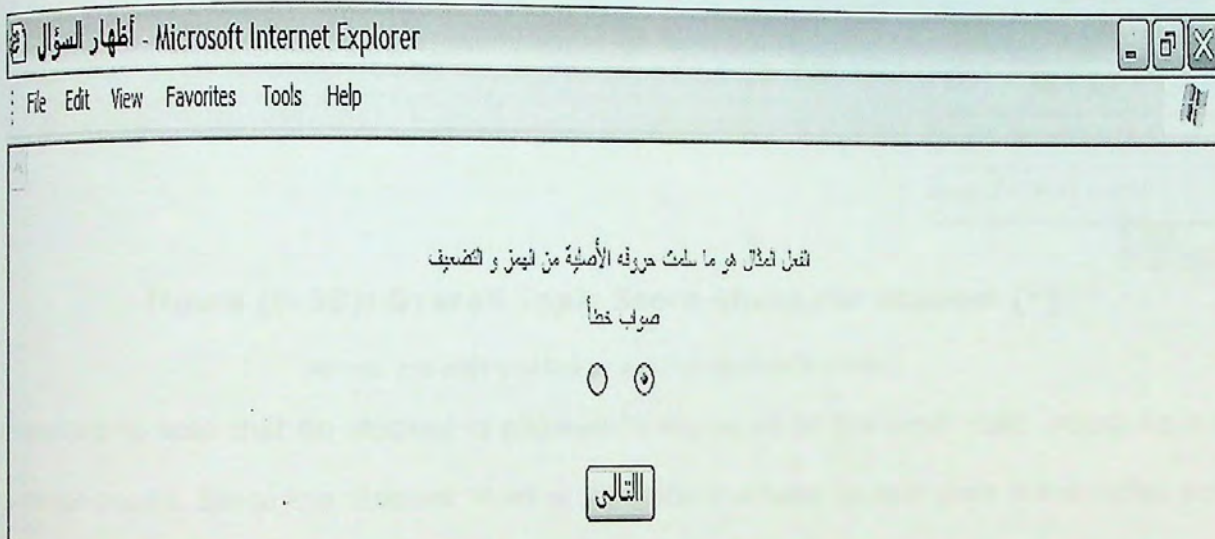


Figure (5-36): Wrong Student Answer on a True/False Question

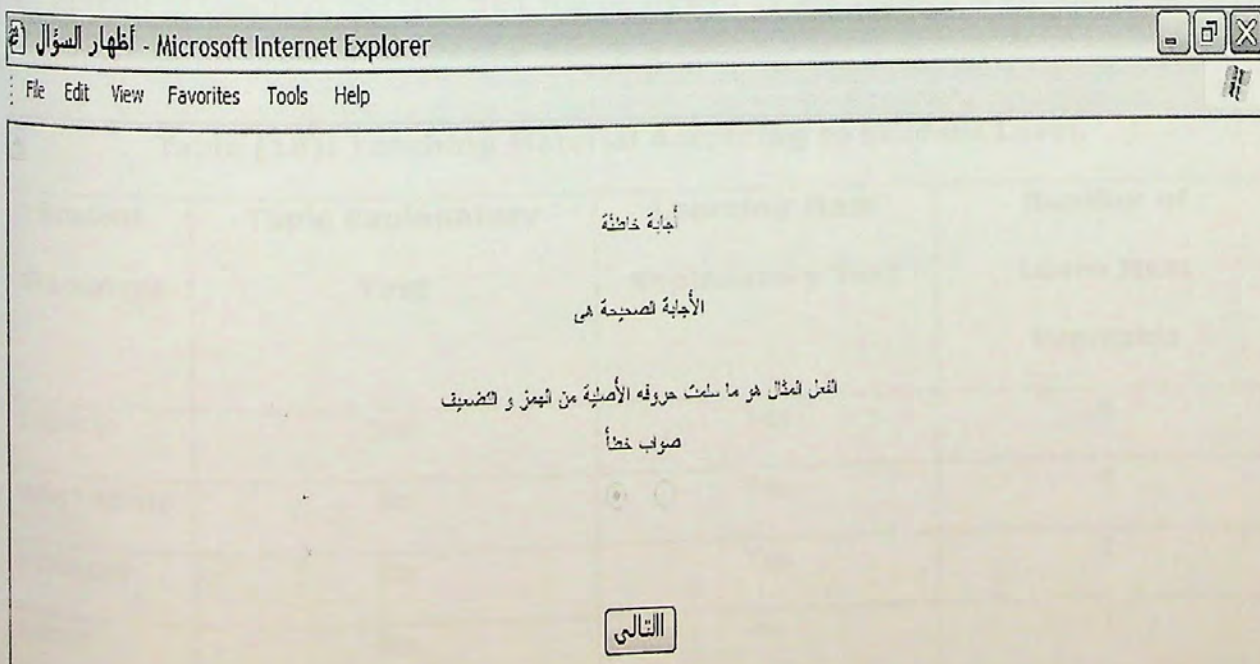


Figure (5-37): Tailored Feedback for True/False Question

Finally, the student answers are graded and he is entitled to improve his category once more as shown in figure (5-38):

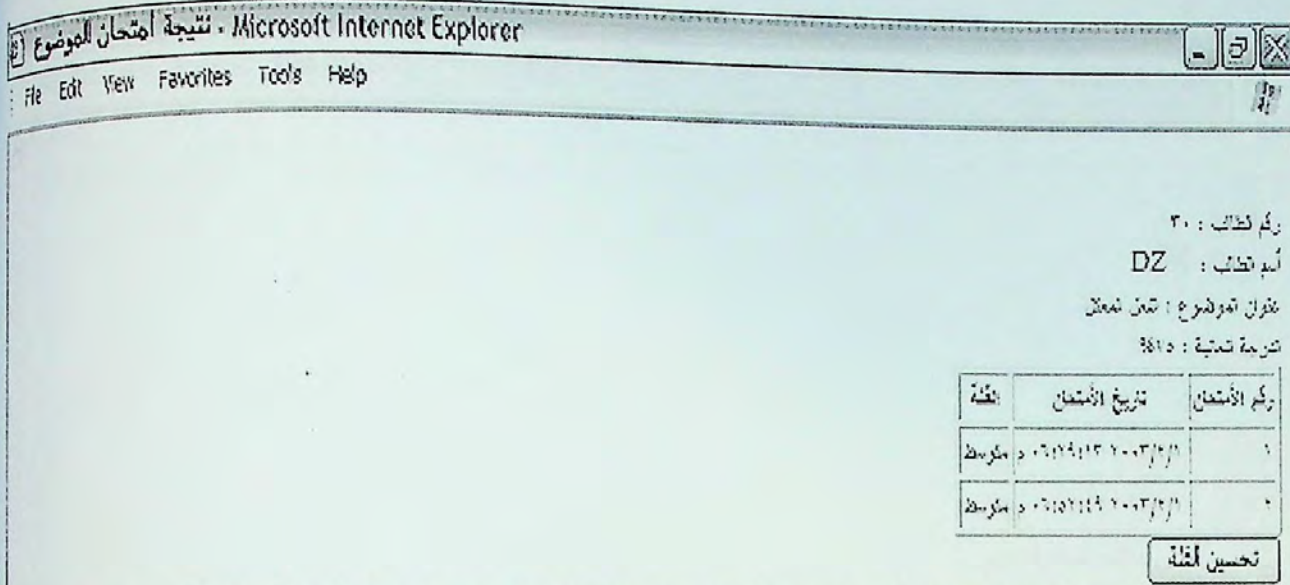


Figure (5-38): Overall Topic Score Sheet Per Student (\*)

Names are abbreviated to protect student's privacy

It is important to note that no student is allowed to move on to the next topic unless he is either advanced or expert. Since the student level is still intermediate he will view 4 examples per learn item, however, the examples that he will view are not the same examples that he viewed earlier. The type of teaching material that will be viewed by the student is illustrated in table (18).

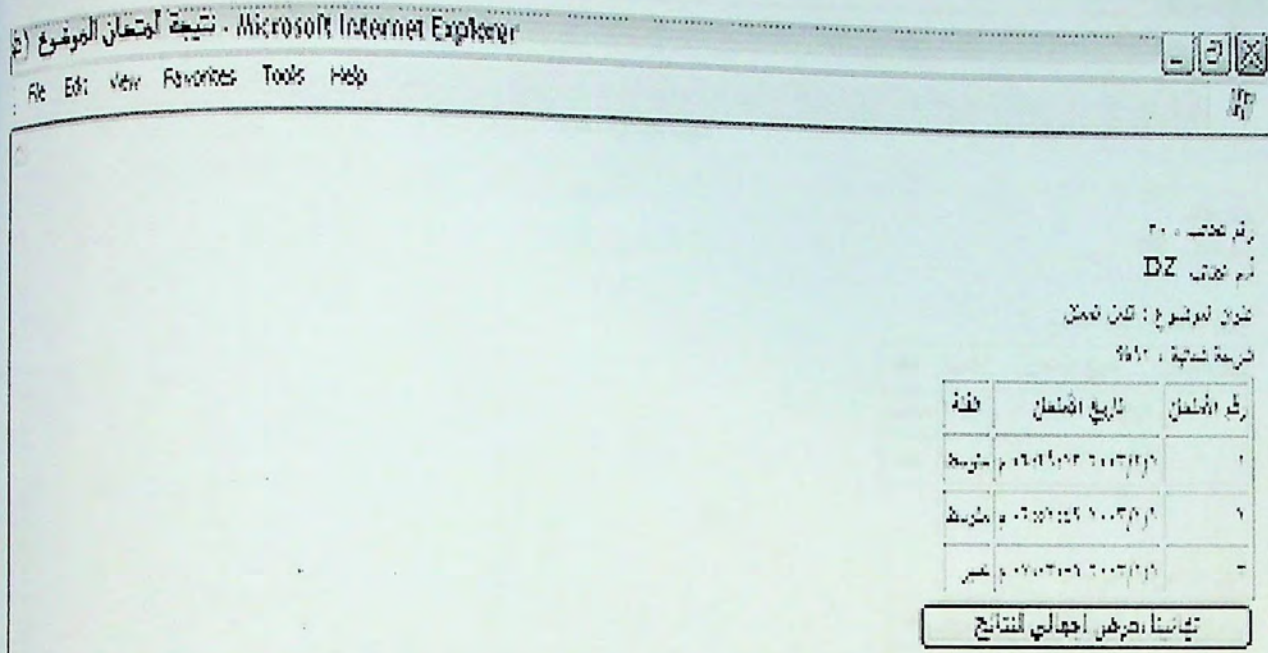
Table (18): Teaching Material According to Student Level.

Student Stereotype	Topic Explanatory Text	Learning Item Explanatory Text	Number of Learn Item Examples
Beginner	Yes	Yes	6
Intermediate	No	Yes	4
Advanced	No	Yes	2
Expert	No	No	1

After the student finishes studying the teaching material he will be subjected to the per topic exam for topic 2. Finally, when the student finishes the per topic exam, he can view the overall



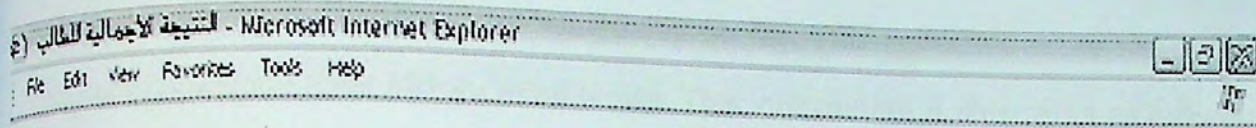
topic score sheet that conveys his progress starting from the entry exam till the final exam that he took. This score sheet is illustrated in figure (5-39):



**Figure (5-39): Final Topic Score Sheet Per Student (\*)**

Names are abbreviated to protect students' privacy

After the student finishes learning all the topics he will be presented by an overall topics score sheet as shown in figure (5-40).



تقييمية الأحيائية تكتب

رقم طالب ٢٠  
 اسم تكتب BZ

عنوان الموضوع : التل فسيج

رقم الامتحان	تاريخ الامتحان	نسبة	نقطة
١	٢٠٠٣/٠٢/٠٢	١٠٠	١٠٠
٢	٢٠٠٣/٠٢/٠٢	١٠٠	١٠٠

عنوان الموضوع : التل فسيج

رقم الامتحان	تاريخ الامتحان	نسبة	نقطة
١	٢٠٠٣/٠٢/٠٢	١٠٠	١٠٠
٢	٢٠٠٣/٠٢/٠٢	١٠٠	١٠٠
٣	٢٠٠٣/٠٢/٠٢	١٠٠	١٠٠

Figure (5-40): Overall Topics Score Sheet Per Student (\*)

Names are abbreviated to protect students' privacy

As illustrated in figures (5-39) and (5-40) each student has two types of score sheets:

- **Overall Topics Score Sheet:** This is a score sheet that contains the following information :

- **Student ID**
- **Student Name**
- **Topic Title:** Since each student may take a number of exams that is related to the same topic until he achieves mastery so each topic in the score sheet has a number of records that convey the following information:

- Exam ID
- Exam Date
- Grade Percentage
- Student Stereotype Per Topic

- **Overall Topic Scores Sheet:** This is a collective score sheet that contains information about all the topics that the student studied so far. This collective score sheet conveys student knowledge level history in all topics. This information is illustrated as follows:

- **Student ID**
- **Student Name**

For each topic that the student studied through the **Arabic Tutor**, he will view the following information:

- **Topic Title:** Since each student may take a number of exams that is related to the same topic until he achieves mastery so each topic in the score sheet has a number of records that convey the following information:
  - Exam ID
  - Exam Date
  - Grade Percentage
  - Student Stereotype Per Topic

#### **Arabic Tutor GUI Module**

The user interface of the **Arabic Tutor** is divided into two parts:

- **Student Interface**

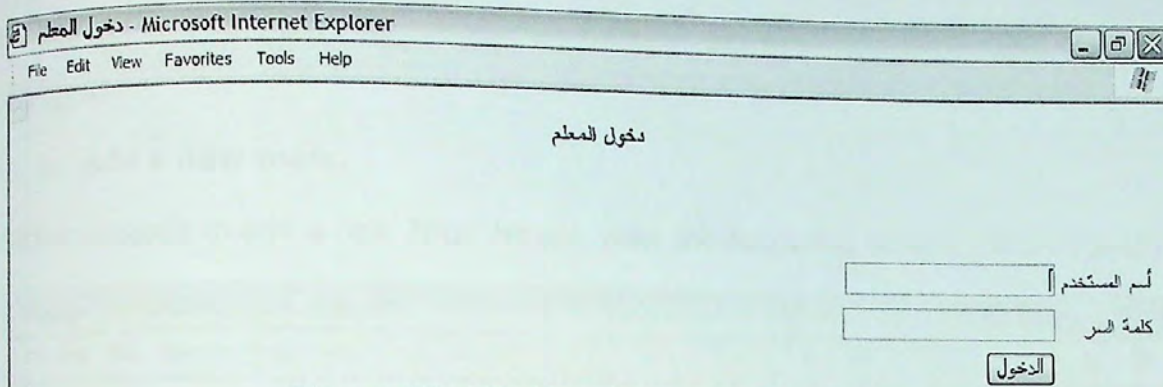
That consists of a set of dynamically constructed HTML pages and forms. The decision about the content and the form of each page is made by the tutoring component.

- **Teacher Interface**

That represents the interface that is viewed by the teachers. This interface provides the teacher with the following:

1. Easy interface to create teaching material this teaching material represent the domain knowledge that will be used by the tutor module later on. We provided the instructors with appropriate tools for creating a course and customizing it according to their preferences.
2. Access to Students' Progress Reports.

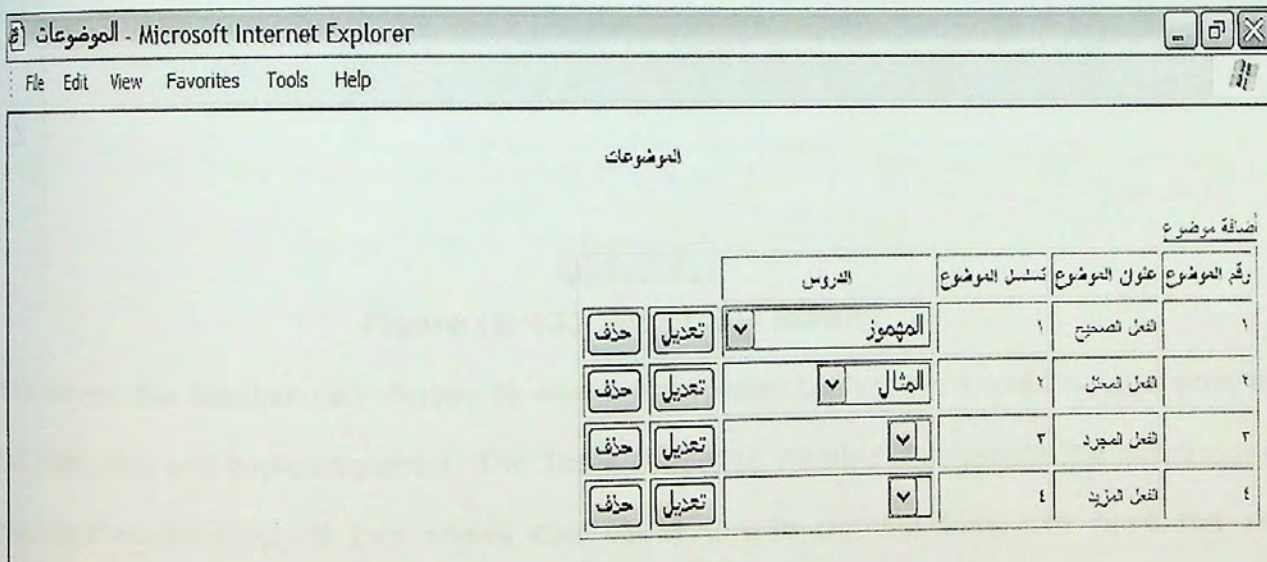
The first step for the teacher to use the **Arabic Tutor** is to log on through the following screen (figure (5-41)):



**Figure (5-41): Teacher Login Screen**

#### 5.4.17 Course Creation and Customization Pattern in the Arabic Tutor

After the teacher logs onto the system he/she will view the following screen (figure (5-42)):



**Figure (5-42): Topics Screen**

The previous screen contains a list of the available topics.

Each topic has the following information:

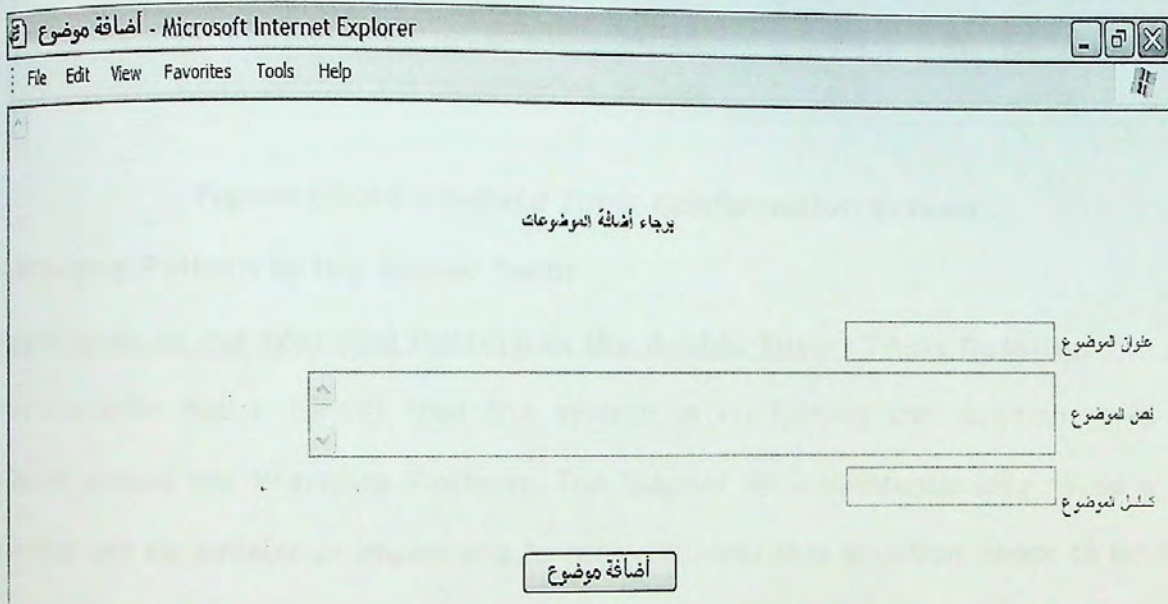
- Topic Id
- Topic Title
- Topic Sequence
- Learn Items Title

In addition the instructor can choose to do any of the following:

1. Add a new topic.
2. Delete a topic.
3. Edit a topic.

**1. Add a new topic.**

If the teacher chooses to add a new Topic he will view the following screen (figure (5-43)):



**Figure (5-43): Add Topic Screen**

In this screen the teacher can choose to add a new topic. Each topic should have a title, topic explanatory text and topic sequence. The Topic sequence number represents the successor and predecessor relationship. If two topics can be taught in parallel they can have the same sequence number. Moreover, it will be used by the tutor module to determine the order of presenting this topic to the student according to its sequence and the student's current knowledge level. The topic sequence helps in achieving student goals that are explained in the

**User Goals Pattern.**

**2. Delete a topic.**

If the teacher chooses to delete a topic he will view the following screen (figure (5-44)):

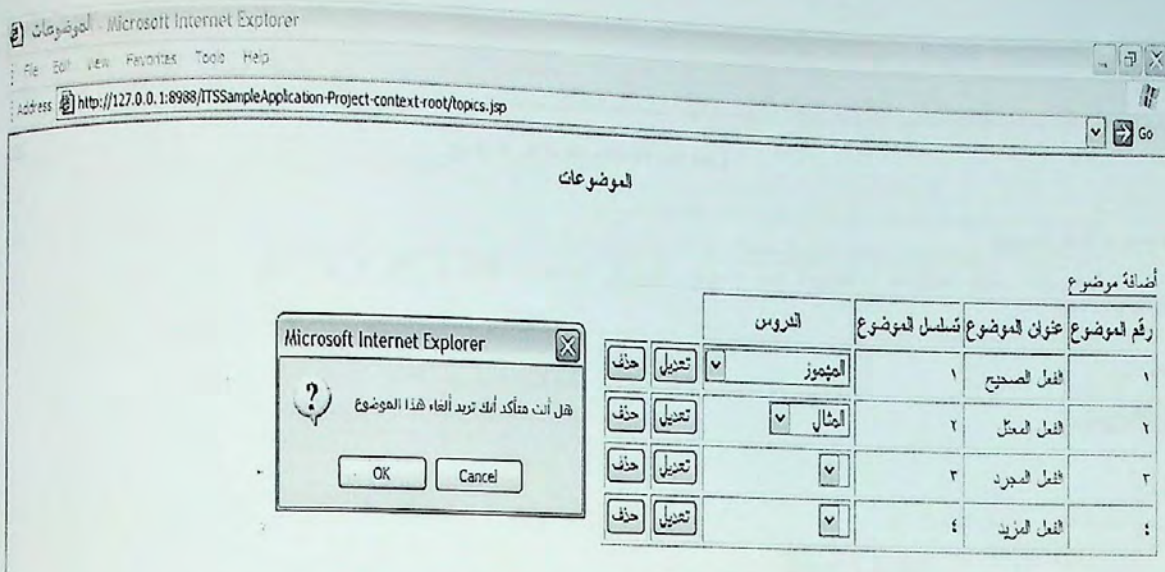


Figure (5-44): Delete Topic Confirmation Screen

#### 5.4.18 Warning Pattern in the Arabic Tutor

##### First Application of the Warning Pattern in the Arabic Tutor: Topic Deletion

It is obvious from figure (5-44) that the system is confirming the deletion. This deletion confirmation utilizes the **Warning Pattern**. The teacher may unintentionally cause a problem situation that will be difficult or impossible to recover from; this situation needs to be resolved. If this action is fully completed it may lead to loss of work. However, the system should not automatically resolve this situation instead; the user needs to be consulted. Therefore, the system should explain to the users why an action could be damaging so as to understand the consequences or options.

In the previous screen we show a message that warn the user before continuing the task and give the user the chance to abort the tasks. This will lead to decreasing the number of user mistakes.

#### 3. Edit a topic.

If the teacher chooses to edit a topic he/she will view the following screen (figure (5-45)):

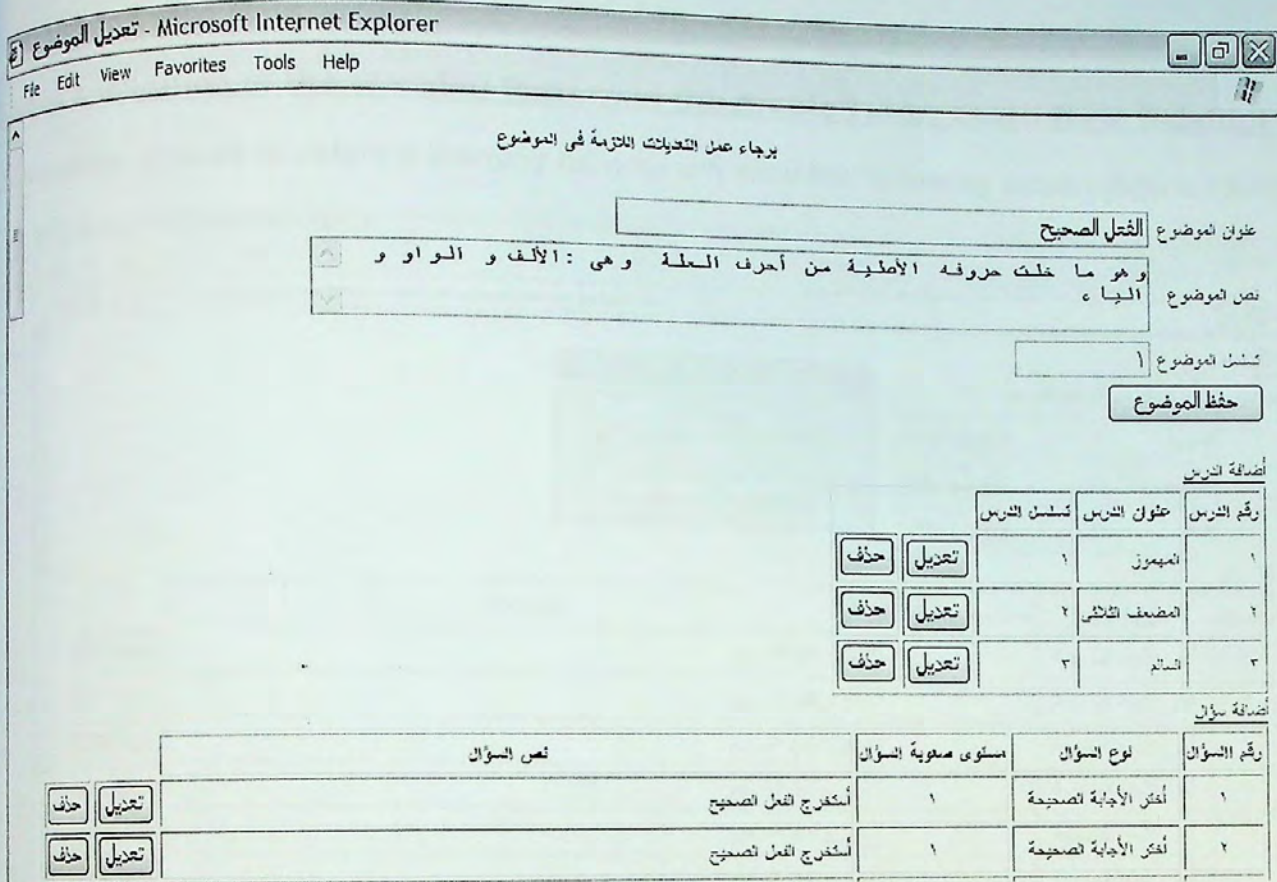


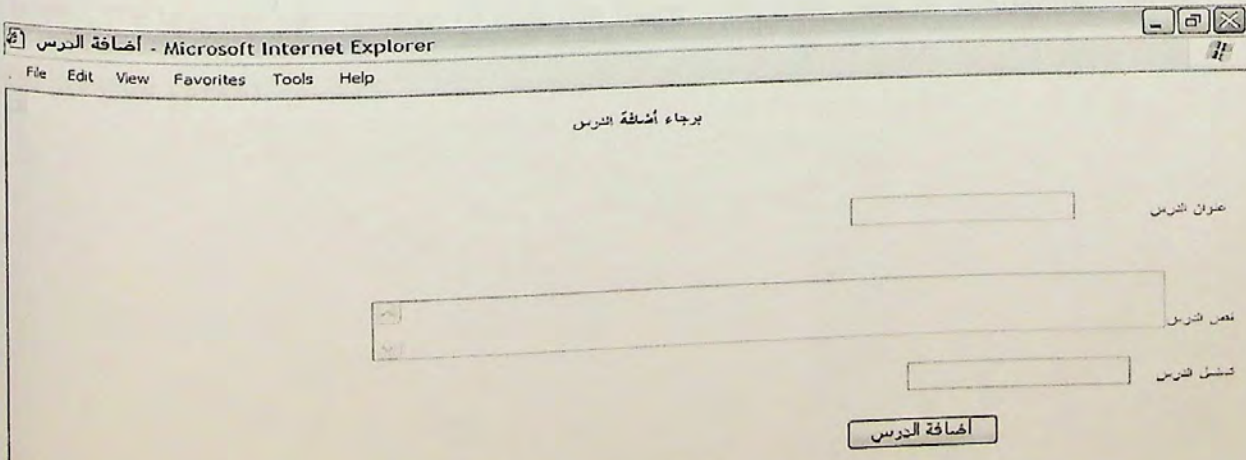
Figure (5-45): Edit Topic Screen

In the Edit Topic window the teacher can do the following:

1. **Add Topic Related information:** Topic title, topic explanatory text, sequence.
2. **Add Learn Item Related Information:** Learn Item ID, title, sequence.

The Add Learn Item Related Information screen is illustrated in figure (5-46).

The item sequence helps in achieving student goals that are explained in the **User Goals** Pattern.



### 3. Delete Learn Item.

#### Second Application of the Warning Pattern in the Arabic Tutor: Learn Item Deletion

If the teacher chooses to delete a learning item he will view the following screen (figure (5-47)):

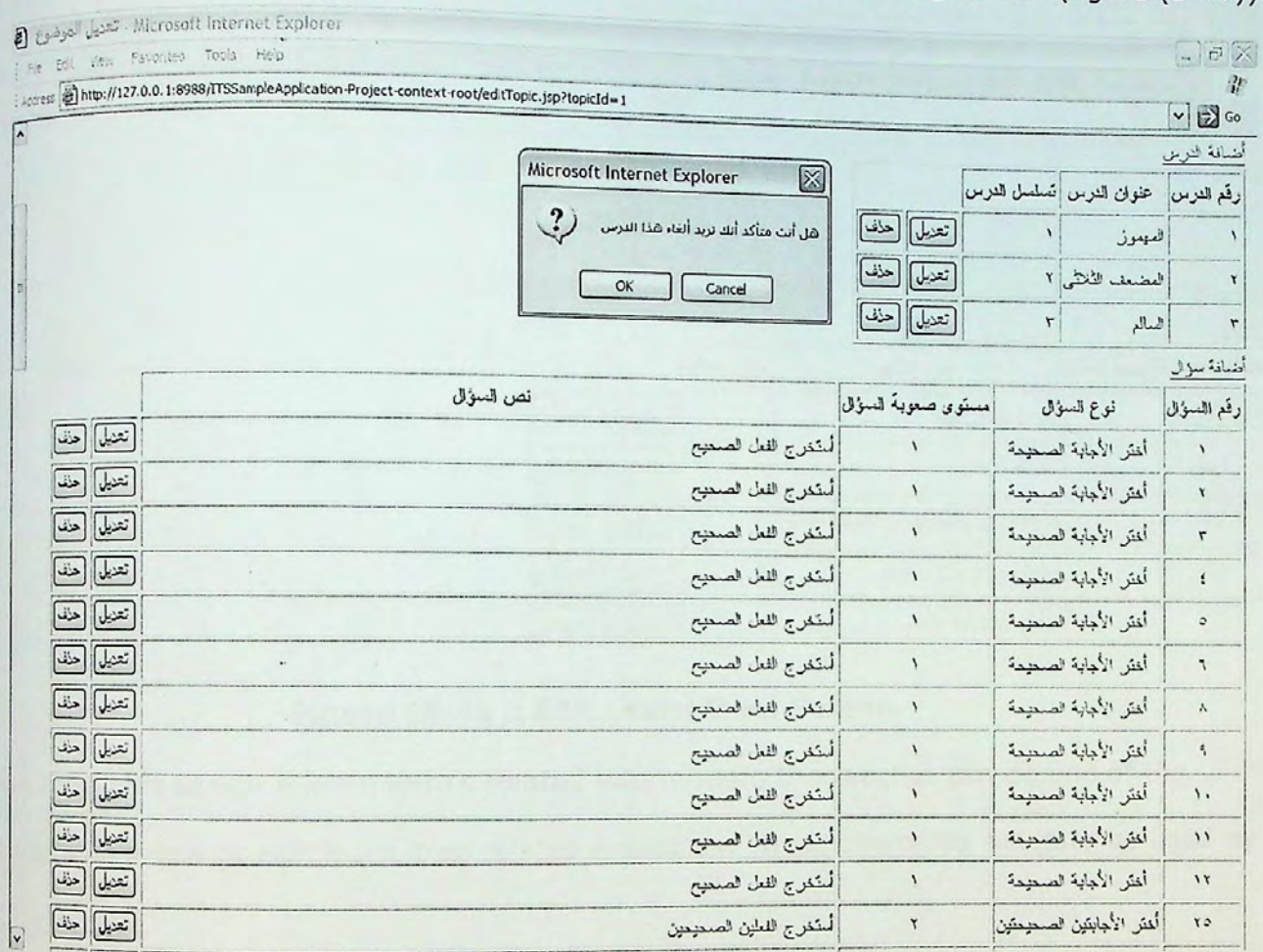


Figure (5-47): Delete Learn Item Confirmation Screen

### 4. Edit Learn Item

In this screen, the teacher can choose to edit the learn item title, text or sequence. This is illustrated in figure (5-48):



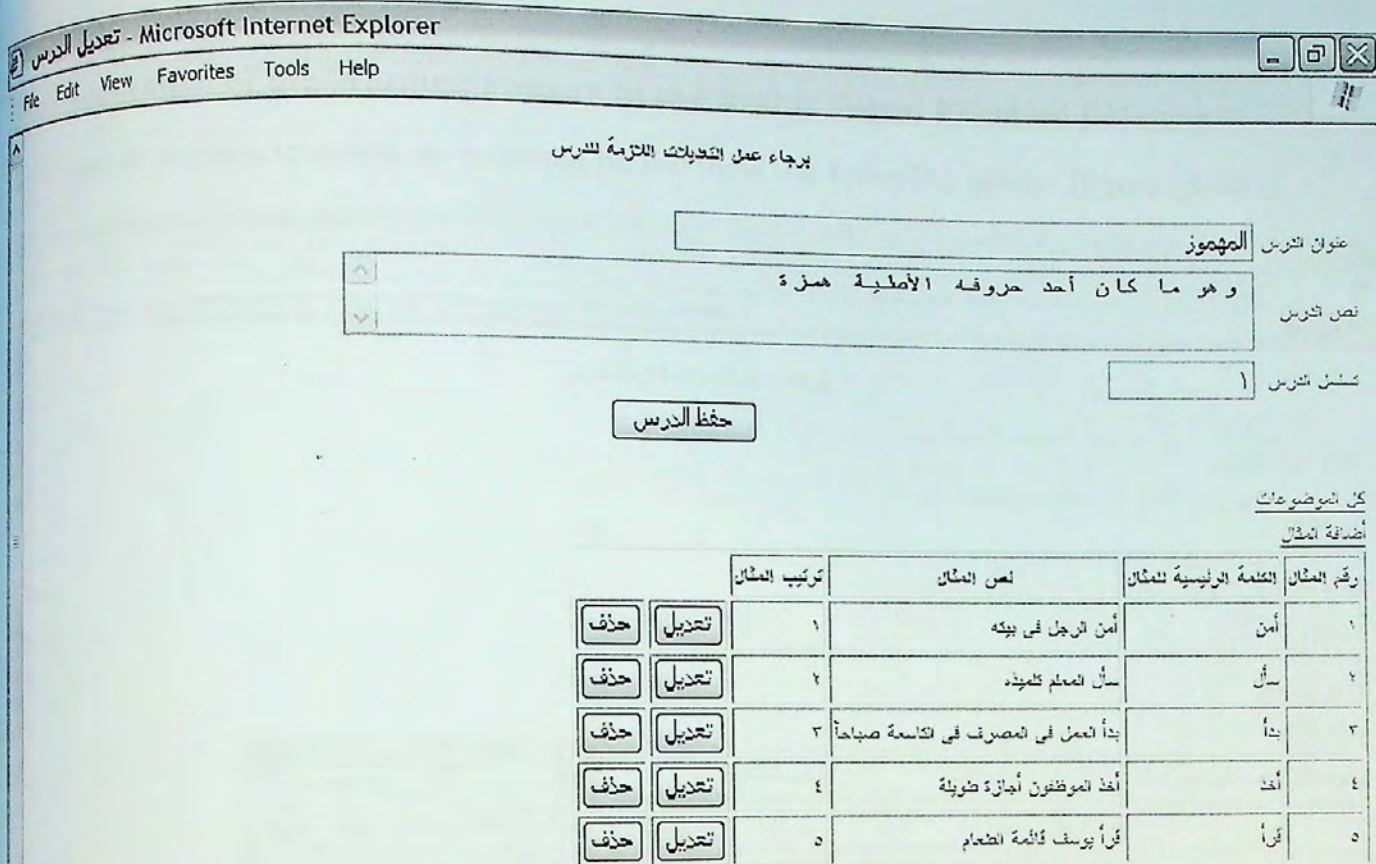


Figure (5-48): Edit Learn Item Screen

Besides being able to edit a learn item's related information, the teacher can do one of the following: add, delete or edit learn item related examples. These examples will later be used by the tutor module.

### 5. Add Example

In addition, each learn item has a set of related examples. The edit learn item screen can allow the teacher to add learn item related examples. Each example has an example keyword, text and sequence. This is illustrated in figure (5-49):

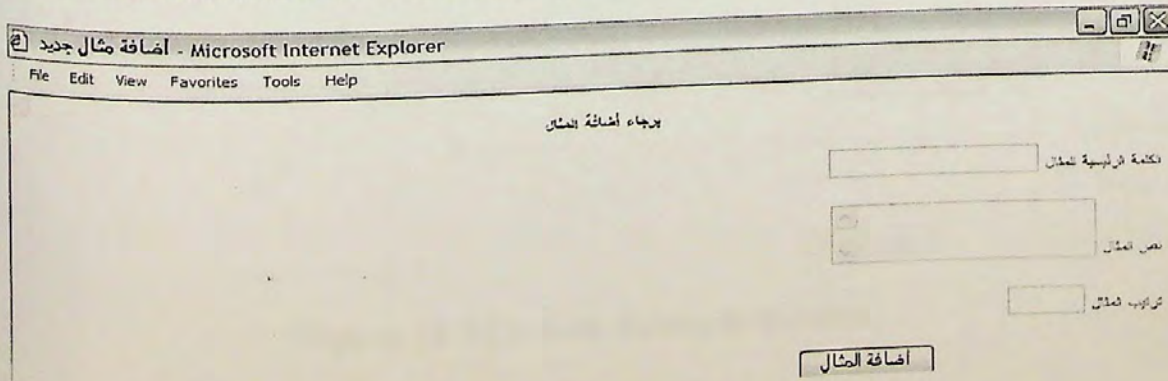


Figure (5-49): Add Example Screen

6. Delete Example

Third Application of the Warning Pattern in the Arabic Tutor: Example Deletion

If the teacher chooses to delete an example he will view the following screen (figure (5-50)):

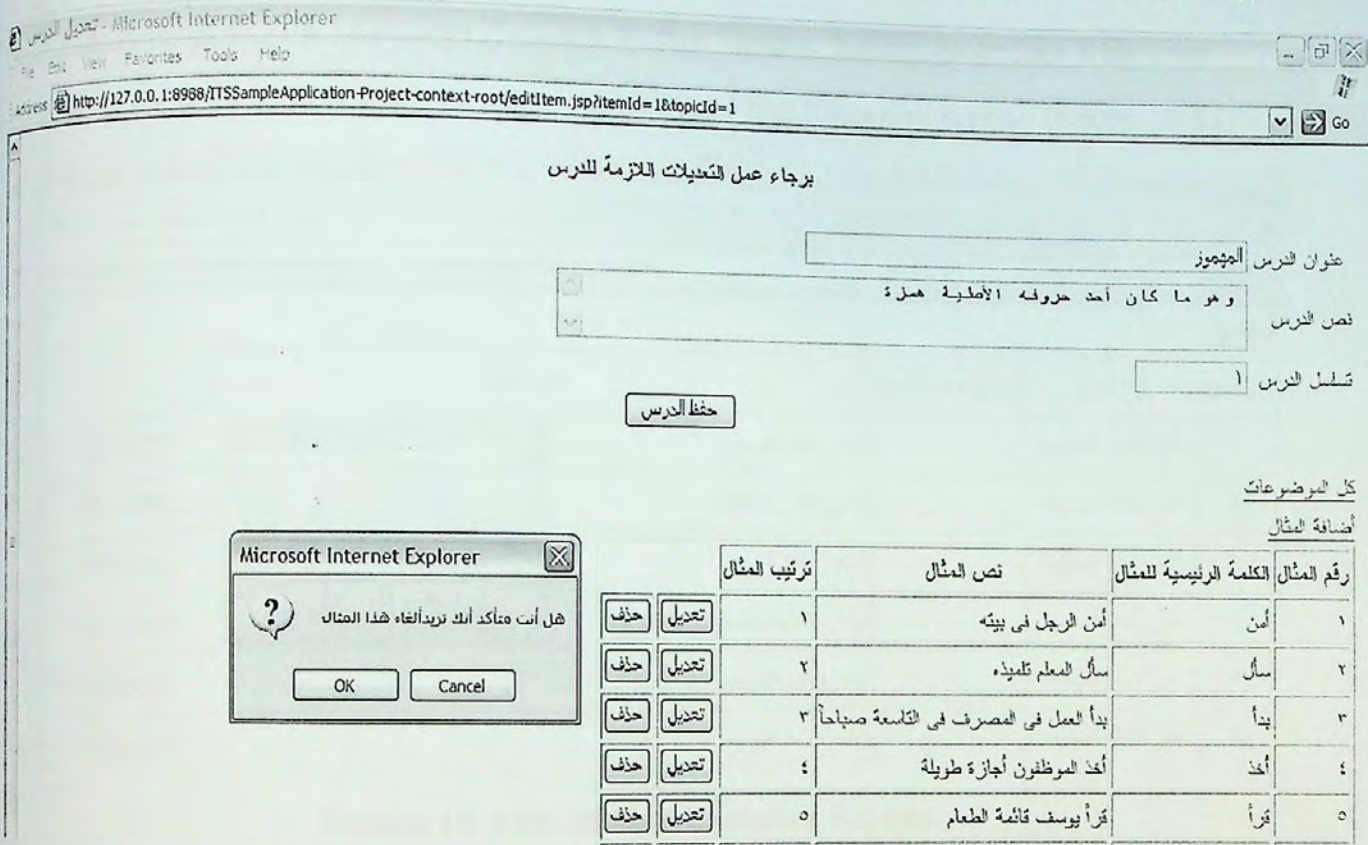


Figure (5-50): Delete Example Confirmation Screen

7. Edit Example

The teacher can also use the system to edit any added example as shown in figure (5-51):

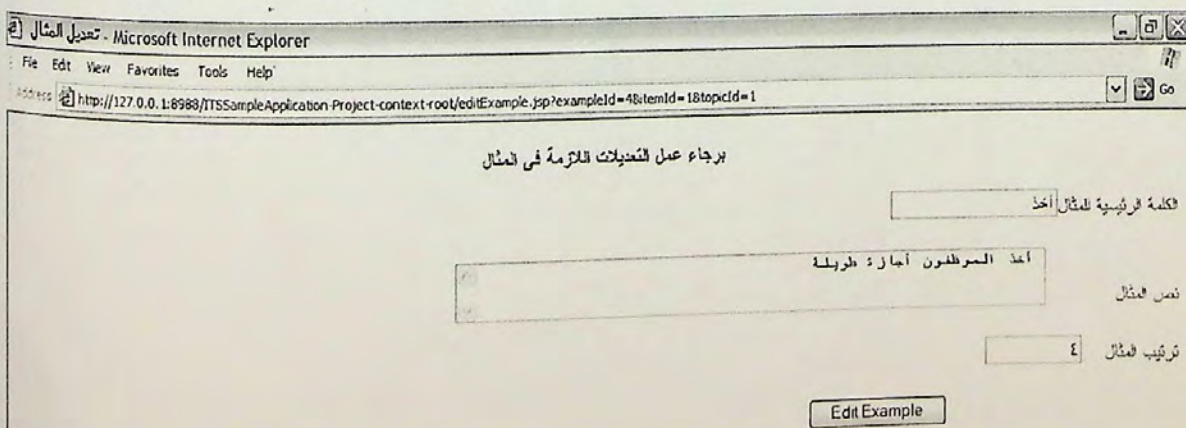


Figure (5-51): Edit Example Screen

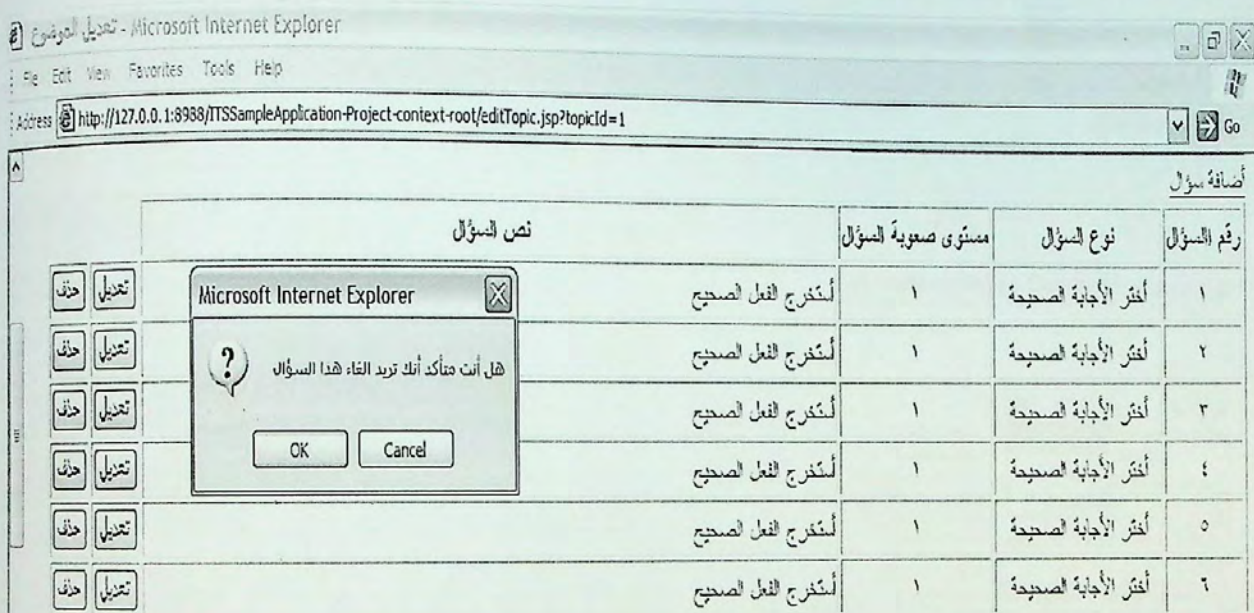
8. Add Exercise Related Information: Which includes:

- Exercise Type: Multiple choice questions, yes/no question, enter verb.

- **Exercise Difficulty Level:** There are 3 difficulty levels available.
- **Exercise Text:**
- **Learn Item(s) that are Covered by the Exercise:**

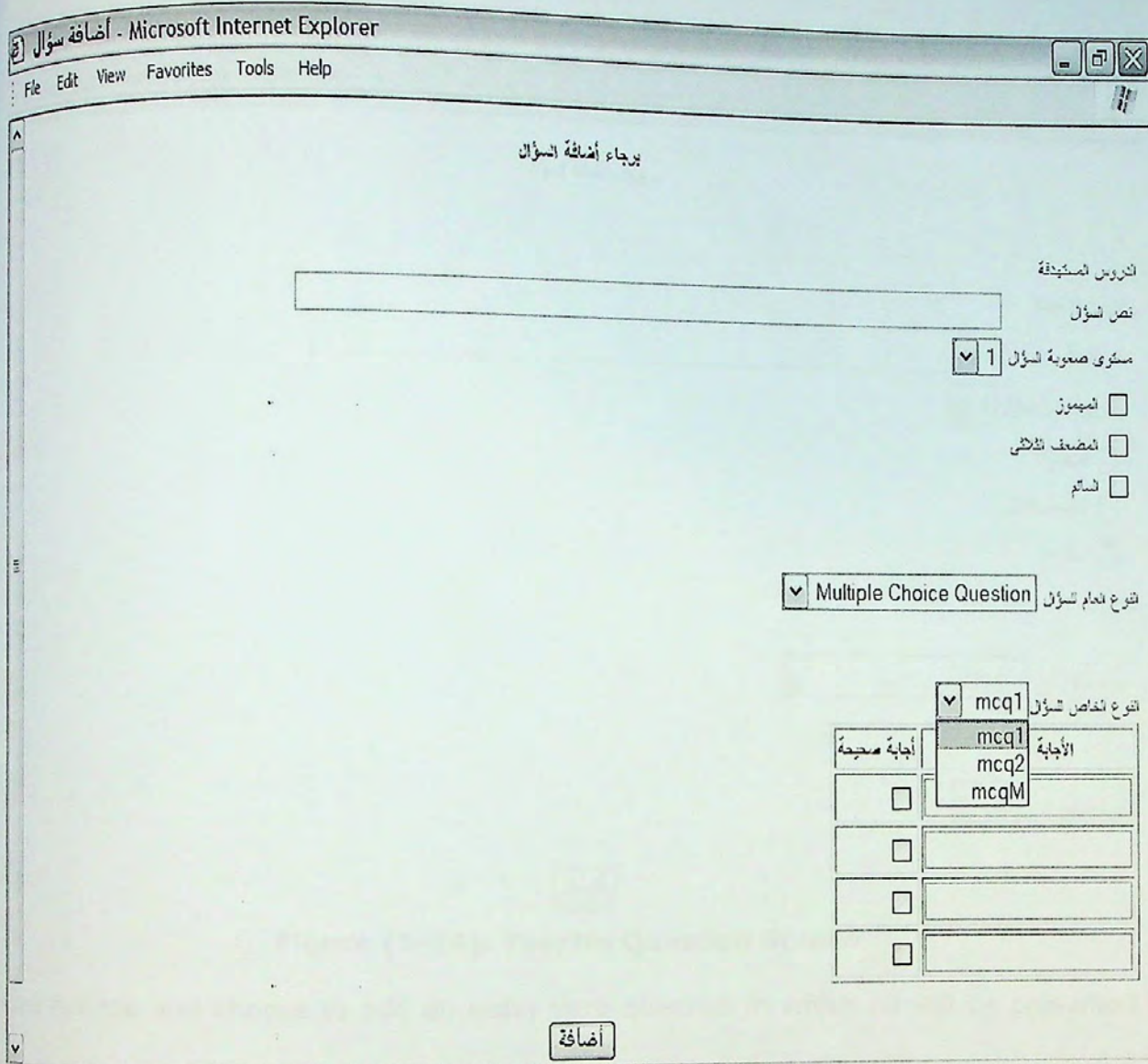
**Fourth Application of the Warning Pattern in the Arabic Tutor: Exercise Deletion**

If the teacher chooses to delete an exercise he will view the following screen (figure (5-52)):



**Figure (5-52): Delete Exercise Screen**

If the teacher wants to add a multiple choice question he will be presented with the following screen. This screen allows him to enter the question text, difficulty level, the learn item(s) that are covered by this question and question type. If the question type was multiple choices then the teacher will have to enter the choices and state which one(s) are correct.



**Figure (5-53): Add Multiple Choice Question Screen**

If the teacher chooses to add a yes/no question, the screen will be changed and he will view the following screen. This screen allows him to enter the question text, difficulty level, the learn item(s) that are covered by this question and question type.

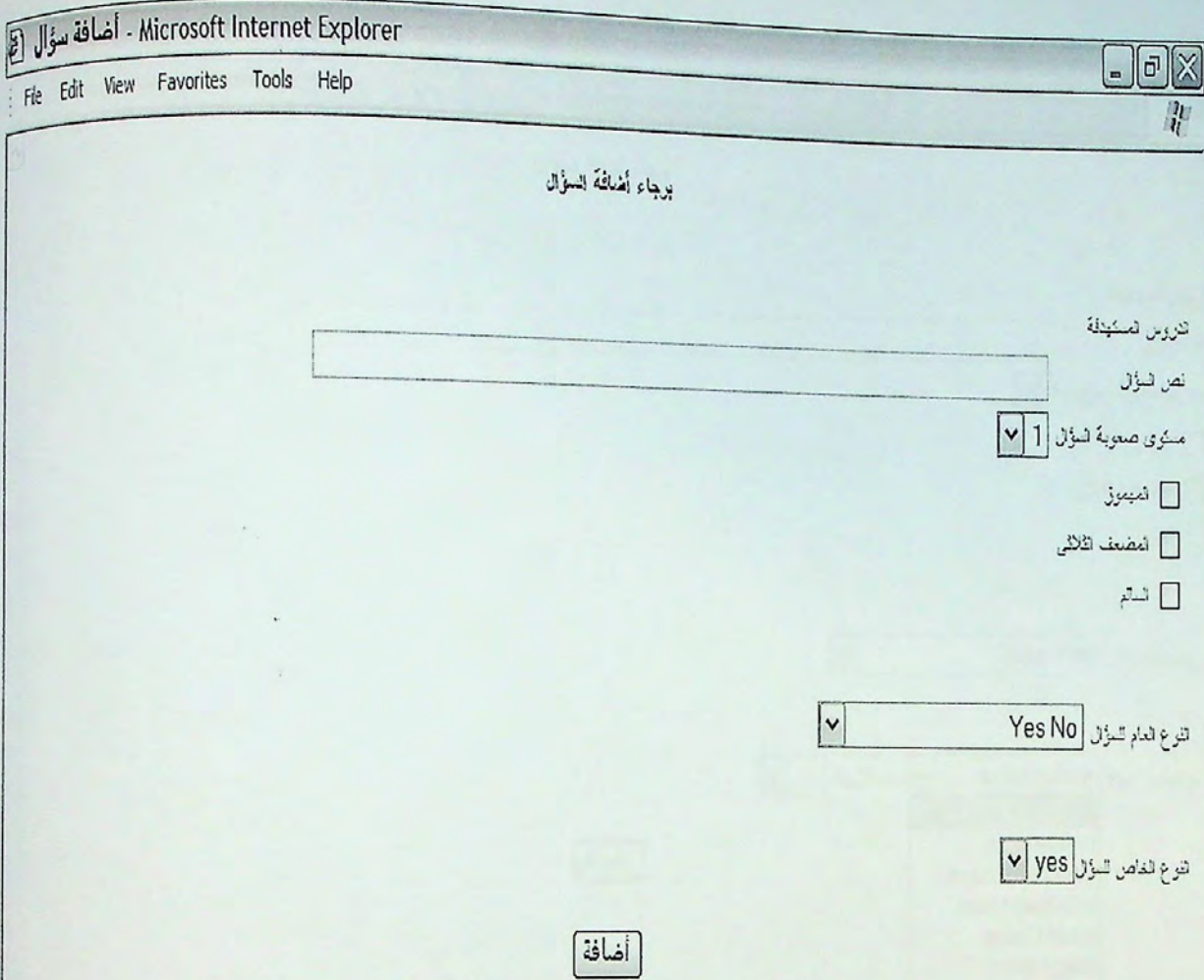


Figure (5-54): Yes/No Question Screen

The teacher can also choose to add an enter verb question in which he will be presented by a screen that has the following options:

- o Enter Saheh
- o Enter salem
- o Enter mahmooz
- o Enter moda3af
- o Enter mesal
- o Enter agwaf
- o Enter naqes
- o Enter mo3tal

This screen allows him to enter the question text, difficulty level, the learn item(s) that are covered by this question and question type and subtype.

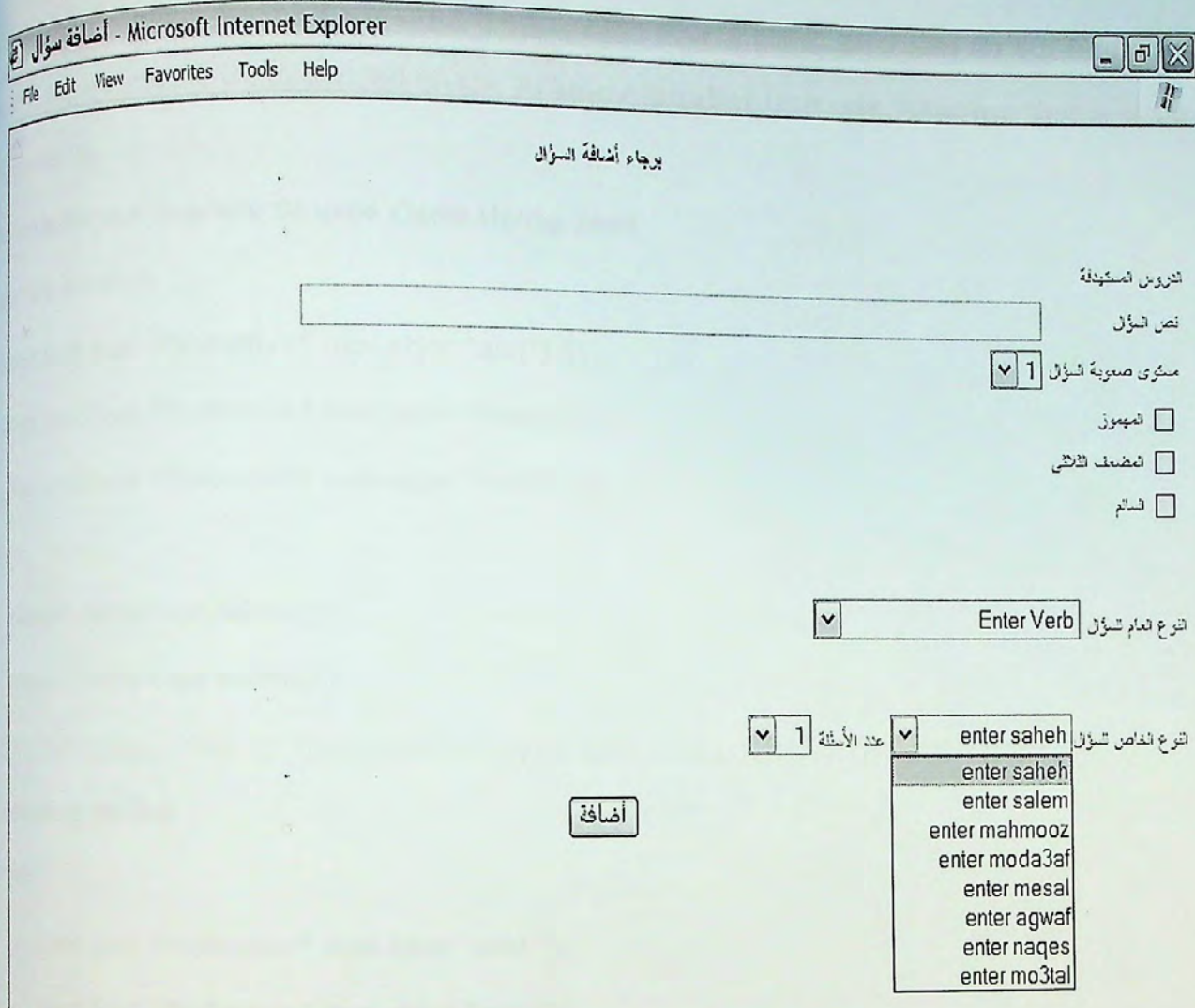


Figure (5-55): Enter verb Question Screen

The enter verb questions are considered to be one of the strongest features in the **Arabic Tutor**. Students are presented with a question that asks them to type a verb with specific features. The student entry is passed to the **Arabic Tutor expert module** that uses Jess (Java Expert System Shell) as a rule-based inference engine. Student's entry is checked against the expert module rules.

In our expert system instead of using the Unicode of these letters and shapes inside our application thus leading to non readable code we used an English name for every Unicode so as to improve the readability of the expert module. A sample of the expert module source code is shown below.

Please refer to appendix G for sample expert system source code using Jess for our future work. In our expert module we depended on the **Arabic Alphabet Unicode Mapping** that is listed in Appendix B.

**Sample Expert module Source Code Using Jess**

```
(defrule is-saheh
  (test (not (call ?*infinite* hasLetter "alef")) )
  (test (not (call ?*infinite* hasLetter "waw")) )
  (test (not (call ?*infinite* hasLetter "yeh")) )
  =>
  (assert (verb-type saheh))
  (assert (verb-type salem)))

.....

(defrule is-mo3tal
  (or
    (test (call ?*infinite* hasLetter "alef"))
    (test (call ?*infinite* hasLetter "waw"))
    (test (call ?*infinite* hasLetter "yeh"))
  ) =>(assert (verb-type mo3tal)))

.....

(defrule is-mahmouz
  (verb-type saheh)
  (test (call ?*infinite* hasLetter "hamza_on_alef"))
  ?x <-(verb-type salem)
  =>(assert (verb-type mahmouz))
  (retract ?x))

.....

(defrule is-moda3af
  (verb-type saheh)
```

```
(test (eq (call ?*infinitive* getPositionOfShaddah) 2))
```

```
(test (eq (call ?*infinitive* getNumberOfCharacters) 3))
```

```
?x <-(verb-type salem)
```

```
=>(assert (verb-type moda3af))
```

```
(retract? x))
```

```
.....
```

```
(defrule is-mesal
```

```
(verb-type mo3tal)
```

```
(or
```

```
(test (eq (call ?*infinitive* findLetter "alef") 1))
```

```
(test (eq (call ?*infinitive* findLetter "waw") 1))
```

```
(test (eq (call ?*infinitive* findLetter "yeh") 1))
```

```
)=>(assert (verb-type mesal)))
```

```
.....
```

```
(defrule is-agwaf
```

```
(verb-type mo3tal)
```

```
(or
```

```
(test (eq (call ?*infinitive* findLetter "alef") 2))
```

```
(test (eq (call ?*infinitive* findLetter "waw") 2))
```

```
(test (eq (call ?*infinitive* findLetter "yeh") 2))
```

```
)=>(assert (verb-type agwaf)))
```

```
.....
```

```
(defrule is-naqes
```

```
(verb-type mo3tal)
```

```
(or
```

```
(test (eq (call ?*infinitive* findLetter "alef") 3))
```

```
(test (eq (call ?*infinitive* findLetter "waw") 3))
```

```
(test (eq (call ?*infinitive* findLetter "yeh") 3))
```



)=>(assert (verb-type naqes)))

#### Fourth Application of the Singleton Pattern in the Arabic Tutor: Collective Student Information Instantiation

In the **Arabic Tutor**, the **CollectiveStudentInfo** is a class that is responsible for generating all statistical data about students. This information is used in generating student progress reports. Since we needed to have only one report generator we used the **Singleton Pattern** to ensure that this class has only one instance, and provide a global point of access to it.

The **Singleton Pattern Structure in the Arabic Tutor** is illustrated in figure (5-56):

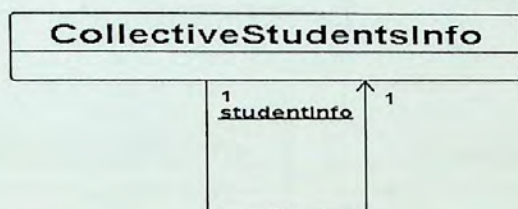


Figure (5-56): Singleton Pattern in the **CollectiveStudentInfo** Class in the **Arabic Tutor**

#### 5.4.19 Observer Pattern in the Arabic Tutor

In the **Arabic Tutor**, in order to observe the status of all students and update the teacher reports with the new student status as he navigates through the learning path we needed the **Observer Pattern**. **Observer Pattern** defines a one to many dependencies between objects so that when one object changes state, all its dependents are notified and updated automatically [7]. Thus if the teacher is monitoring the status of all students he will always see the up-to-date results of all students even if a student has just taken an exam.

In order to implement the **Observer Pattern** we followed the following steps:

1. **Creating the Subject Participant:** This is represented by the **Observable** interface. That provides an interface for attaching and detaching **Observer** objects. The **Subject Participant** knows its **Observers**. Any number of **Observer** objects may observe a subject.
2. **Creating the Observer Participant.** This is represented by the **Observer** interface. This **Participant** defines an updating interface for objects that should be notified of changes in

a subject.

3. **Creating the Concrete Subject Participant.** This is represented by the Student Class. This participant stores state of interest to Concrete Observer objects and sends a notification to its Observers when its state changes.
4. **Creating the Concrete Observer Participant.** This is represented by the CollectiveStudentInfo class. This participant maintains a reference to Concrete Subject object and stores state that should stay consistent with the subject's and implements the Observer updating interface to keep its state consistent with the subject's.

The Observer Pattern Structure in the Arabic Tutor is illustrated in figure (5-57):

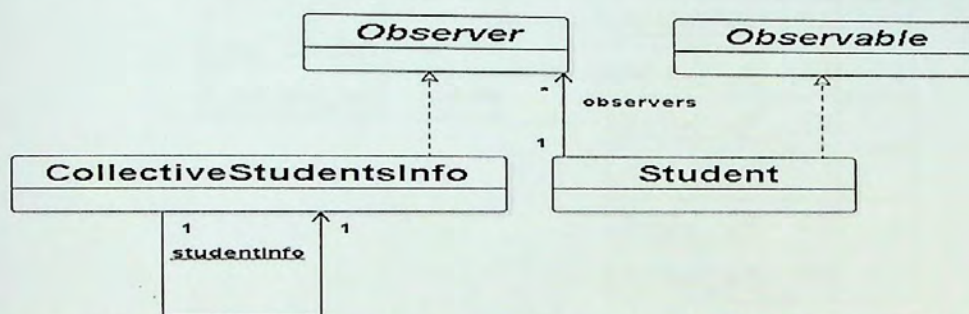
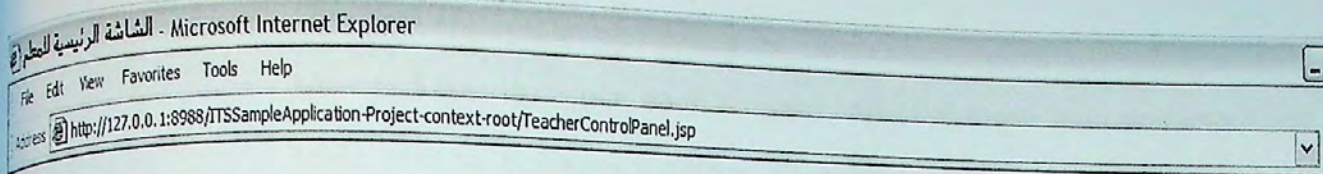


Figure (5-57): Observer Pattern Structure in the Arabic Tutor

In the Arabic Tutor the following steps take place:

1. The CollectiveStudentsInfo class implements the Observer interface.
2. Student class implements Observable interface.
3. Whenever a student object is changed it sends an update () message to all Observers (i.e. CollectiveStudentsInfo) so CollectiveStudentsInfo is always up-to-date and instantly reflects the changes.

After the teacher logs into the **Arabic Tutor** he views the following screen:



الشاشة الرئيسية للمعلم

رقم الموضوع	عنوان الموضوع	النتيجة الإجمالية للموضوع	رقم الطالب	اسم الطالب	النتيجة الإجمالية للطالب
١	الفعل الصحيح	النتيجة الإجمالية للموضوع	MS		النتيجة الإجمالية للطالب
٢	الفعل المعتل	النتيجة الإجمالية للموضوع	AO		النتيجة الإجمالية للطالب
٣	الفعل المجرد	النتيجة الإجمالية للموضوع	AA		النتيجة الإجمالية للطالب
٤	الفعل المزيد	النتيجة الإجمالية للموضوع	NM		النتيجة الإجمالية للطالب
			NM		النتيجة الإجمالية للطالب
			NA		النتيجة الإجمالية للطالب
			BA		النتيجة الإجمالية للطالب
			ZM		النتيجة الإجمالية للطالب
			RF		النتيجة الإجمالية للطالب
			SM		النتيجة الإجمالية للطالب
			DM		النتيجة الإجمالية للطالب

Figure (5-58): Main Reports Screen (\*)

Names are abbreviated to protect students' privacy

Figure (5-58) illustrates the Main reports screen that is divided into two parts:

- **Part One:** Contains the following information:
  - Student ID
  - Student Name
  - Link to **Overall Student Grade in All Topics**
- **Part Two:** Contains the following information:
  - Topic ID
  - Topic Title
  - Link to **Overall Students Grade in this Topic**

The teacher may choose to view one of the following reports:

1. **Collective Topic Progress Report.** This report contains the following information:
  - **Topic Title:** Each topic has a the following list of information:

- **Student ID**
- **Student Name:** Since each student can take more than one exam per topic until he achieves mastery, each student has the following information that is related to his status in each topic:
  - **Exam Date**
  - **Grade Percentage**
  - **Student Stereotype**

This report is illustrated in figure (5-59):

Category	Percentage	Date	Student Name	Student ID
			MS	1
مبتدی	55%	2003/10/20	AO	2
متقدم	55%	2003/10/20	AA	3
مبتدی	55%	2003/10/20	NM	4
متقدم	55%	2003/10/20		
مبتدی	55%	2003/10/20	NM	5
متوسط	55%	2003/10/20		
متقدم	55%	2003/10/20	NA	6
مبتدی	55%	2003/10/20	BA	7
متوسط	55%	2003/10/20	ZM	8
متقدم	55%	2003/10/20	RF	9
مبتدی	55%	2003/10/20		
متوسط	55%	2003/10/20	SM	10
متقدم	55%	2003/10/20	DM	11
مبتدی	55%	2003/10/20		

[back](#)  
**Figure (5-59): Collective Topic Progress Report (\*)**

Names are abbreviated to protect students' privacy

In addition the teacher can choose to view the report that shows each individual student progress in all learned topic.

## 2. Student Progress in All Topics Report.

This is a collective score sheet that contains information about all the topics that the student studied so far. This collective score sheet conveys student knowledge level history in all topics.

This information is illustrated as follows:

- **Student ID**
- **Student Name**

For each topic that the student studied through the **Arabic Tutor**, he will view the following information:

- **Topic Title:** Since each student may take a number of exams that is related to the same topic until he achieves mastery so each topic in the score sheet has a number of records that convey the following information:
  - Exam ID
  - Exam Date
  - Grade Percentage
  - Student Stereotype Per Topic



النتيجة الإجمالية للطالب

رقم الطالب ٦٠  
اسم الطالب N.M

عنوان الموضوع : علم التسميح

رقم الامتحان	تاريخ الامتحان	النسبة	الدرجة
١	٢٠٠٣/١٠/٣١ - ٢٠٠٣/١١/١٠	55.4	ممتاز
٢	٢٠٠٣/١٠/٣١ - ٢٠٠٣/١١/٣٧	56.00	ممتاز

عنوان الموضوع : علم التمثيل

رقم الامتحان	تاريخ الامتحان	النسبة	الدرجة
١	٢٠٠٣/١٠/٣١ - ٢٠٠٣/١١/١٠	55.6	ممتاز
٢	٢٠٠٣/١٠/٣١ - ٢٠٠٣/١١/٣٧	56.00	ممتاز

Figure (5-60): Student Progress in All Topics Report

### 5.5 The Arabic Tutor Class Diagram

Figure (5-61) represents a collective class diagram for the **Arabic Tutor** and lists all the classes involved in its implementation. These classes were discussed in detail in section 5.4; however the importance of this diagram is that it represents a comprehensive overview to all the classes integrated together.

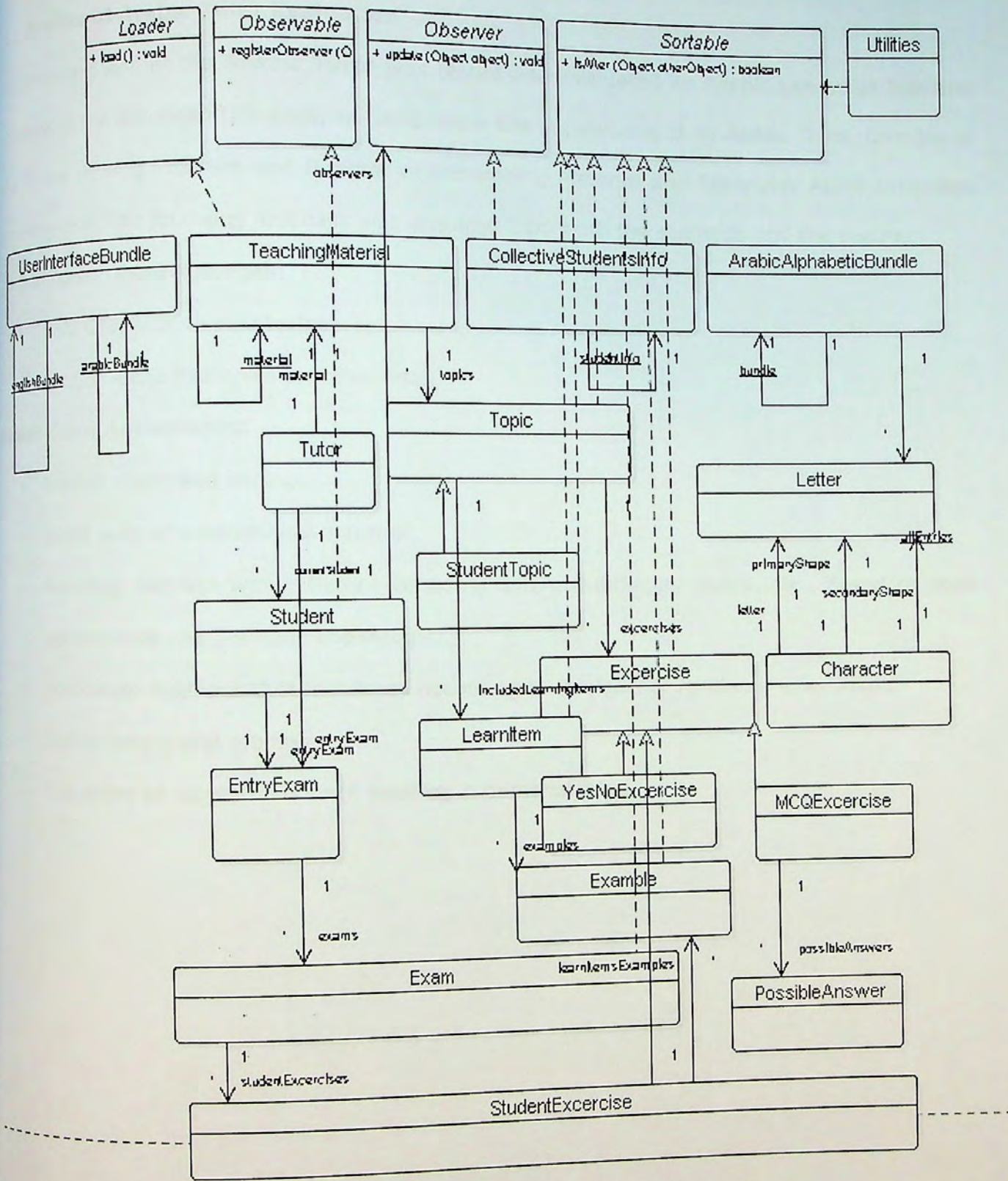


Figure (5-61): The Arabic Tutor Class Diagram

### 5.6 Results of Arabic Tutor Evaluation

As mentioned earlier the **Arabic Tutor** was tested and evaluated on Arabic Language Institute students in the American University in Cairo under the supervision of Mr. Abbas Tonsi, Director of the Arabic Writing Program and Director of Developing Material and Computer Aided Language Learning Unit. The following feedback was acquired from both the students and the teacher:

#### **Arabic Tutor Disadvantages:**

- Poor Graphical User Interface.
- Lack of Audio Examples and Exercises.

#### **Arabic Tutor Advantages:**

- Learner controlled pacing.
- Small units of instructional material.
- Providing learners with different versions, different difficulty levels, etc., based on their performance and previous knowledge.
- Immediate feedback that is tailored according to the learner level and error made.
- Online testing and grading.
- The ability to retake tests until mastery is demonstrated.



## Chapter 6

### Summary and Conclusion

The design and implementation of Intelligent Tutoring Systems (ITS) is a very complex task, as it involves a variety of organizational, administrative, instructional and technological components. In addition, there are no well established methodologies or development tools for ITS implementation. Therefore systematic, disciplined approaches must be devised in order to leverage the complexity of ITS implementation and achieve overall product quality within specific time and budget limits.

ITS complexity can be overcome by creating a pattern language for intelligent tutoring systems that can help software developers resolve recurring problems encountered throughout all of software development process of any ITS. In this way, designers of new or existing ITSs, especially inexperienced designers, can take advantage of previous design expertise and save precious time and resources.

#### 6.1 Achievements and Contributions

##### 6.1.1 Pattern Discovery

In this research we showed that one cannot talk of patterns in the ITS domain only in the context of ITS architectures. On the contrary, there are many kinds of other patterns that can be helpful in ITS implementation.

This research started out by surveying the existing Intelligent Tutoring Systems and their components in search for some possible common design decisions, common interactions among components, and common generalized principles underlying superficially different designs. We extracted patterns from numerous known examples, systems, architectures, designs, learning and teaching styles, strategies, etc.

We investigated the use of design patterns, access patterns, instructional patterns, adaptive patterns, pedagogical patterns and interaction Patterns in ITS implementation and showed that some of them were found to be useful in ITS implementation by building a pattern language called **PLITS** and applying it in developing an Arabic Intelligent Language Tutoring System called the **Arabic Tutor**.

### 6.1.2 Formulating PLITS

**PLITS** is up to our knowledge the first pattern language for intelligent tutoring systems implementation (**PLITS**). It includes rules and guidelines which explain how and when to apply its patterns to solve a problem which is larger than any problem an individual pattern can solve.

**This pattern language aims to achieve the following:**

- Reduce time and cost of designing and developing ITS.
- Increase software quality of the ITSs usability.
- Increase pedagogical quality of the ITSs learning effectiveness.

### 6.1.3 Implementing the Arabic Tutor

The **Arabic Tutor** is an Intelligent Tutoring System over the World Wide Web for teaching a subset of the basic rules of the Arabic language that combines the flexibility and intelligence of Intelligent Tutoring Systems with the availability of the World Wide Web applications. We selected the domain of language learning because it is among the few domains that gained international popularity because of its importance to students internationally, irrespective of their mother tongue. The **Arabic Tutor** is our proof of concept prototype. It covers both the areas of intelligent language tutoring systems and patterns for intelligent language tutoring systems in order to prove that our suggested pattern language is useful and can act as a guideline for ITS implementation.

### 6.2 Directions for Future Research

**Suggested directions for future research include the following:**

1. Establishing an initiative for constructing a repository of patterns for intelligent tutoring systems in order to attract more researchers to deposit their own patterns. That would strengthen the pattern language and offer a wealthy pool of patterns, so that

inexperienced designers of an ITS could base their work on a sound and systematic basis.

2. Increasing the scope of the **Arabic Tutor** domain module to include a wider range of topics. Please refer to appendix G for sample expert system source code using Jess for our future work.

3. Introducing pedagogical agents into the **Arabic Tutor** implementation. These pedagogical agents can help us in turning the **Arabic Tutor** into a collaborative ILTS by introducing agents that can have the following roles:

a. **Learning Companion:** which learns to perform the same learning task as the student, at about the same level, and can exchange ideas with the student when presented the same learning material.

b. **Trouble Maker :** which tries to disturb the student by proposing solutions that are at times correct, but are wrong at other times, thus challenging the student's self-confidence in learning.

4. Implementing an ITS Authoring Tool.

An ITS authoring tool is an interactive tool designed to allow teachers and instructors with little technological knowledge to create simple Intelligent Tutoring Systems on the web. Its objective is to deliver the courseware adaptively to meet the needs of different learners. It makes the creation of an ontology describing the content model for a given course much easier. It's an ITS shell along with a user interface that allows non-programmers to formalize and visualize their knowledge [36].

5. Building an ITS Framework.

A framework dictates the architecture and design decisions that are common to an application domain. It will define the overall structure, its partitioning into classes and objects, the key responsibilities thereof, how the classes and objects collaborate, and the thread of control. A framework predefines these design parameters so that the application designer or developer can concentrate on the specifics of his/her application. Frameworks thus emphasize design reuse over code reuse. A typical framework contains several design patterns.

One of the advantages of frameworks over design patterns is that they can be written down in programming languages and executed and reused directly unlike design patterns which have to be implemented each time they are used [8].

6. Introducing Web Services into ITS implementation.

Web Services are self-contained, modular applications that provide a set of functionalities to anyone that requests them. The main characteristic of Web Services is that they interact with the applications that invoke them, using web standards such as WSDL (Web Service Definition Language), SOAP (Simple Object Access Protocol) and UDDI (Universal Description, Discovery and Integration). Basing learner modeling on web standards has the advantage of enabling the dynamic integration of applications distributed over the Internet, independently of their underlying platforms [37].

Web services can be used to provide ITS expert modules for instance to anyone who requests them.

## Bibliography

- [1] Schmidt, D., Fayad, M., Johnson, R.E. Software Patterns, Communications of The ACM, Vol.39, No.10, pp. 37-39,1996.
- [2] Brad Appleton. Patterns and Software: Essential Concepts and Terminology,2000.  
<http://www.enteract.com/~bradapp/docs/patterns-intro.html>. View Date May 2004.
- [3] Liegle,J.,Gyun Woo,H.Developing Adaptive Intelligent Tutoring Systems:A General Framework and Its Implementations.In The Proceedings of ISECON 2000,Vol.17 (Philadelphia).  
<http://isedj.org/isecon/2000/915/ISECON.2000.Liegle.pdf>. View Date May 2004.
- [4] Vladan Devedzic, Andreas Harrer. Common Patterns in ITS Architectures. KI 18(3), Pages 17-21, 2004.
- [5] Antonija Mitrovic. An Intelligent SQL Tutor on the Web.International Journal of Artificial Intelligence in Education, 2003.
- [6] Institute of Electrical and Electronics Engineers. IEEE Standard Computer Dictionary: A Compilation of IEEE Standard Computer Glossaries. New York, NY: 1990.
- [7] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides. Design Patterns, Elements of Reusable Object-Oriented Software, Addison Wesley Professional. First Edition. 1995.
- [8] D. Riehle, H. Zllighoven. Understanding and Using Patterns in Software Development. In: K. Lieberherr, R. Zicari (eds.): Theory and Practice of Object Systems, Special Issue Patterns. Guest Editor: S. Berczuk,Vol. 2, No.1, 1996, pp. 3-13.  
[http://www.ubilab.org/publications/print\\_versions/pdf/tapos-96-survey.pdf](http://www.ubilab.org/publications/print_versions/pdf/tapos-96-survey.pdf). View Date May 2004.
- [9] Alexander, C. The Timeless Way of Building. Oxford University Press,New York,1979.
- [10] James O. Coplien, Patterns Definitions section of the Patterns Home Page,  
<http://hillside.net/patterns/definition.html>. View Date May 2004.
- [11] Andrew Koenig, Why C++ is Not Just an Object Oriented Programming Language,  
Conference on Object Oriented Programming Systems Languages and Applications,1995.  
<http://www.research.att.com/~bs/oopsla.pdf>. View Date May 2004.
- [12] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, Peter

Sommerlad. Pattern-Oriented Software Architecture: A System of Patterns. John Wiley & Sons, 2000.

[13] John Borchers. A Pattern Approach to Interaction Design. John Wiley & Sons, Inc. New York, 2001

[14] M. van Welie, G.C. van der Veer. Pattern Languages in Interaction Design: Structure and Organization. In Proceedings of Interact '03, 1-5 September, Zurich, Switzerland, Eds: Rauterberg, Menozzi, Wesson, p527-534, IOS Press, Amsterdam, The Netherlands, 2003.

[15] Heift, T. Intelligent Language Tutoring System for Grammar Practice, 2001.

[http://www.spz.tu-darmstadt.de/projekt\\_ejournal/jg-06-2/beitrag/heift2.htm](http://www.spz.tu-darmstadt.de/projekt_ejournal/jg-06-2/beitrag/heift2.htm). View Date May 2004.

[16] El-Sheikh, E., and Sticklen, J. A Framework for Developing Intelligent Tutoring Systems Incorporating Reusability. IEA-98-AIE: 11th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, Benicassim, Castellon, Spain, Springer-Verlag, 1998.

[17] M. Virvou & V. Tsigira. Web Passive Voice Tutor: an Intelligent Computer Assisted Language Learning System over the WWW. IEEE International Conference on Advanced Learning Technologies pp 131 – 134, Wisconsin, USA, 2001.

[18] Heift, T., Toole, J., McFetridge, P., Popwich, F., Tsiplakou, S. An Interactive Course Support System for Greek. Bourdeau, J. & Heller, R. (eds). Proceedings of ED-MEDIA 00, World Conference on Educational Multimedia, Hypermedia & Telecommunications, Charlottesville, VA: AACE: 394-399, 2000.

[19] Mayo, M., Mitrovic, A., McKenzie, J. CAPIT: An Intelligent Tutoring System for Capitalisation and Punctuation. International Workshop on Advanced Learning Technologies, Palmerston North, New Zealand, 2000.

[20] FUM, D., Giangrandi, P., Tasso, C. Tense Generation in an Intelligent Tutor For Foreign Language Teaching: Some Issues in The design of the Verb Expert. In Proceedings of 4th Conference of the European Chapter of the Association for Computational Linguistics, Pages 124-129, Manchester. Association for Computational Linguistics, 1989.

- [21] Heift, T. An Interactive Intelligent Tutor over the Internet. Otman, T. & Tomak, I. (eds). Proceedings of ED-MEDIA 1998, World Conference on Educational Multimedia, Hypermedia & Telecommunications, Charlottesville, VA: AACE: 556-560, 1998.
- [22] Harrer, A., Devedzic, V. Design and Analysis Patterns in ITS Architectures, in: Kinshuk, Lewis, R., Akahori, K., Kemp, R., Okamoto, T., Henderson, L., Lee, C.-H. (Eds.). Proceedings of the International Conference on Computers in Education 2002 (ICCE 2002), Auckland, New Zealand, pp. 523-527, December 2002.
- [23] Devedžic, V. Inside Intelligent Tutoring Systems - Using Design Patterns. International Journal of Knowledge-Based Intelligent Engineering Systems, Vol.4, No.1, pp. 25-32, January 2000.
- [24] Devedzic, V. A Pattern Language for Architectures of Intelligent Tutors. In Proceedings of AIED'01. pp 542-544, 2001.
- [25] R. Nkambou and Claire IsaBelle. "Cyberphysique: A Web-based distance learning environment". In: Global Education on the Net, Vol. 1, pp.252-256. Springer-Verlag (Berlin) (1998).
- [26] Heift, T., Toole, J., McFetridge, P., Popwich, F., Tsiplakou, S. Learning Greek with an Intelligent and Adaptive Hypermedia System. IMEJ of Computer-Enhanced Learning, Volume 2(2), October 2000.
- [27] P.Avgeriou, A.Papasalouros, S.Retalis. Patterns For Designing Learning Management Systems. European Pattern Languages of Programming (EuroPLOP), 25th-29th June 2003, Irsee, Germany.
- [28] ELEN Project, <http://www2.tisip.no/E-LEN/outcomes.php>. View Date May 2004.
- [29] <http://www.answers.com/topic/pedagogical-patterns>. View Date May 2004.
- [30] M. van Welie, H. Trætteberg. Interaction Patterns in User Interfaces. 7th. Pattern Languages of Programs Conference, 13-16, Allerton Park Monticello, Illinois, USA, August 2000.
- [31] Bergin, Joseph. Fourteen Pedagogical Patterns. Proceedings of the Fifth European Conference on Pattern Languages of Programs, July 2000, Irsee, Germany.
- [32] Devedzic, V., Jerinic, L., & Radovic, D. The GET-BITS Model of Intelligent Tutoring Systems.

Journal of Interactive Learning Research 11(3), pp 411-434, 2001.

[33] T. Murray. Authoring Knowledge Based Tutors: Tools for Content, Instructional Strategy, Student Model, and Interface Design. Journal of the Learning Sciences. Journal of the Learning Sciences, 7(1), 5-64, 1998.

[34] W. Van Joolingen, S. King, and T. De Jong. The SimQuest Authoring System for Simulation-Based Discovery Learning. In du Boulay, B., Mizoguchi, R. (eds.). Artificial Intelligence in Education. IOS Press, Amsterdam / OHM Ohmsha, Tokyo, pp. 79-86, 1997.

[35] P. Avgeriou, A. Papasalouros, S. Retalis, E. Skordalakis. Towards a Pattern Language for Learning Management Systems. Educational Technology & Society, ISSN 1436-4522, Volume 6, Issue 2, pp. 11-24, 2003. <http://ifets.ieee.org/periodical/6-2/2.html> . View Date May 2004.

[36] Tom Murray. Authoring Intelligent Tutoring Systems: An Analysis of the State of the Art. International J. of Artificial Intelligence in Education. Volume 10, pp. 98-129, 1999.

[37] Kabassi, K., & Virvou, M. Using Web Services for Personalized Web-based Learning. Educational Technology & Society, 6(3), 61-71, 2003.

[http://ifets.ieee.org/periodical/6\\_3/8.html](http://ifets.ieee.org/periodical/6_3/8.html). View Date May 2005.

[38] Ernest Friedman-Hill, Jess in Action, Java Rule-based Systems, Ernest Friedman-Hill, Manning Publications July 2003.

[39] فؤاد نعمة، ملخص قواعد اللغة العربية، القاهرة ، مطبعة نهضة مصر ، الطبعة السادسة، 1982.

[40] نبيل راغب، القواعد الذهبية لأتقان اللغة العربية، القاهرة، مكتبة غريب، 1986

[41] السعيد محمد البدوي، الكتاب الأساسي في تعليم اللغة العربية لغير الناطقين بها، الطبعة الثانية، المنظمة العربية للتربية والثقافة والعلوم، 1992.

[42] إبراهيم يوسف السيد، القواعد العربية الميسرة، جامعة الملك سعود.

[43] حسين كامل فتوح، الأعراب في النحو و الصرف، الطبعة الأولى، دار البيان للطباعة، القاهرة، 1999.



## Appendix A : The Arabic Tutor Development Environment

1. **JDeveloper 10g**: the development kit that was used. It has a robust testing environment for web applications. It uses the rapid application development (RAD) to build Java applications, JSP pages, applets, and java beans.
2. **Front Page XP**: It will be used to develop HTML pages.
3. **UML**: Was used to depict the design pattern notations. The Unified Modeling Language (UML) is the standard language nowadays for designing, visualizing, documenting object oriented software.
4. **JDK (Java Development Kit) 1.4**: Was used as the implementation language. Java is currently very widespread in the internet community for its network flexibility and its innovative solutions.
5. **JSP 1.2**: Was used to develop the dynamic pages for the web application. Java Server Pages (JSP) are like HTML pages, but they provide dynamic content inside the HTML.
6. **Oracle database version 9.2**
7. **Jess [38]**: Jess is an interpreter for the *Jess rule language*. The syntax of the Jess rule language is similar to that of Lisp; it is well-suited to both defining rules and procedural programming. **Jess has been used to develop a broad range of commercial software, including:**
  - Expert systems that evaluate insurance claims and mortgage applications
  - Agents that predict stock prices and buy and sell securities
  - Network intrusion detectors and security auditors
  - Design assistants that help mechanical engineers
  - Smart network switches for telecommunications
  - Servers to execute business rules
  - Intelligent e-commerce sites
  - Games

## Appendix B: Arabic Alphabet Unicode Mapping

letter.hamza=\u0621	letter.qaf=\u0642
letter.madda_on_alef=\u0622	letter.kaf=\u0643
letter.hamza_on_alef=\u0623	letter.lam=\u0644
letter.hamza_on_waw=\u0624	letter.meem=\u0645
letter.hamza_under_alef=\u0625	letter.noon=\u0646
letter.hamza_on_yeh=\u0626	letter.heh=\u0647
letter.alef=\u0627	letter.waw=\u0648
letter.beh=\u0628	letter.alef_maksura=\u0649
letter.teh_marbuta=\u0629	letter.yeh=\u064A
letter.teh=\u062A	letter.hamzat_wasl_on_alef=\u0671
letter.theh=\u062B	letter.wavy_hamza_on_alef=\u0672
letter.jeem=\u062C	letter.wavy_hamza_under_alef=\u0673
letter.hah=\u062D	letter.high_hamza=\u0674
letter.khah=\u062E	letter.high_hamza_alef=\u0675
letter.dal=\u062F	letter.high_hamza_waw=\u0676
letter.thal=\u0630	letter.hamza_yeh=\u0678
letter.reh=\u0631	shape.fathatam=\u0648
letter.zain=\u0632	shape.dammatan=\u064C
letter.seen=\u0633	shape.kasratan=\u064D
letter.sheen=\u0634	shape.fatha=\u064E
letter.sad=\u0635	shape.damma=\u064F
letter.dad=\u0636	shape.kasra=\u0650
letter.tah=\u0637	shape.shadda=\u0651
letter.zah=\u0638	shape.sukun=\u0652
letter.ain=\u0639	shape.maddah_above=\u0653
letter.ghain=\u063A	letter.feh=\u0641

Appendix C: Arabic Lessons, Examples and exercises [39] [40] [41] [42] [43]

مقدمة

الميزان الصرفي

تختص قواعد الصرف ببنية الكلمة العربية و كل ما يطرأ عليها من تغيير سواء بالزيادة أو بالنقص .

و معظم الكلمات العربية ثلاثية الحروف . و لذا أعتبر علماء الصرف أن أصول الكلمات ثلاثة أحرف ، و وضعوا نظاماً لضبط بنية الكلمة و قابلوها عند وزنها بالفاء و العين و اللام (فعل) .

على هذا الأساس تكون كلمة ( شكر ) على وزن فعل و ( شرب ) على وزن فعل و ( كرم ) على وزن فعل . و إذا كانت الكلمة مزيدة بحرف أو أكثر من حروف الزيادة و هي الحروف التي تجمعها كلمة "سالتمونيتها" قوبلت الحروف الأصلية بالفاء و العين و الهمزة و زيدت في الميزان الحروف الزائدة كما هي بحركاتها .

و على ذلك تكون كلمة أحسن على وزن أفعال، و كلمة شارك على وزن فاعل و كلمة أستكر على وزن استفعال و كلمة كاتب على وزن فاعل، و كلمة محرم على وزن مفعول و كلمة انتخاب على وزن افتعال .

و إذا حذف حرف من الكلمة الموزونة حذف ما يقابله في الميزان . و على هذا تكون كلمة خذ على وزن عل و فعل الأمر ف من وفي على وزن ع :

هذا و تنقسم الكلمة العربية الى ثلاثة أقسام :

اسم - فعل - حرف .

و سندرس في هذا الجزء قواعد الصرف المتعلقة بالفعل .

و سنتناول دراسة الفعل بالنظر الى بنيته و ينقسم الى صحيح و معتل

تقسيم الفعل بالنظر الى بنيته

• و ينقسم الفعل بالنظر الى بنيته الى صحيح و معتل .

الأفعال الصحيحة و المعتلة

أولاً: الفعل الصحيح:

تعريفه: وهو ما خلت حروفه الأصلية من أحرف العلة (الواو، الألف، الياء)

أقسام الفعل الصحيح : و ينقسم الفعل الصحيح ثلاثة أقسام :

• المهموز:

تعريفه: وهو ما كان أحد حروفه الأصلية همزة .

مثل : أمن ، سال ، بدأ ، أخذ ، قرأ ، أكل ، أمر ، سئم ، نشأ ، أضر ، ملأ .

• أمن الرجل في بيته .

• سال المعلم تلميذه .

• بدأ العمل في المصرف في التاسعة صباحاً .

• أخذ الموظفون اجازة طويلة .

• قرأ يوسف قائمة الطعام .

• أكل الشاب الطعام .

• أمر الوالد ولده بالأجتهد .

• سئم الطالب من الفراغ .

• نشأ الطفل في عائلة غنية .

• أضر الطالب الكسول عمل اليوم الى الغد .

• ملأ الرجل الكوب ماءً .

• المضعف الثلاثي :

تعريفه: و هو ما كان وسطه و آخره من جنس واحد، ويمكن أيضا القول أن الفعل المضعف هو ما كان ثاني حروفه الأصلية وثالثها من جنس واحد.

مثل : جف ، هز ، عز ، شد ، رد ، عد ، دق ، مد ، فص ، ضم ، مر ، ظل ، نص ، حل ، شق ، دل ، شم ، بث ، حط ، فر ، ظن ، قل ، حل ، صب ، سد ، تم ، هم .

- هزّ الريح ورقّ الشجر .
- عزّ فراق الأبناء على الأم .
- شدّ الطفل الحبل .
- ردّ الشاب على الهاتف .
- عدّ الموظف النقود .
- دقّ بانج اللين جرس الباب .

• السالم :

تعريفه: وهو ما سلمت حروفه الأصلية من الهمز و التضعيف .  
مثل : عمل ، زرع ، فتح ، كتب ، فهم ، لعب ، رسم ، نقل ، رغب ، رفض ، قدم ،  
عرض ، جلس ، سرق ، صعد ، رجع ، سبّح ، شرب ، درس ، كسر ، حفر ، نظر ، حضر ، خرج ، طبخ ، سكن ، غسل .

- عمل الفلاح بأجتهاد .
- من زرع حصد .
- فتح الولد الباب .
- كتب الطالب البحث .
- فهم التلميذ الدرس .
- لعب فريق الجامعة مباراة قوية .
- رسم الفنان لوحة جميلة .
- نقلت الأقمار الصناعية وقائع الاحتفال .
- رغب الشاب في الزواج .
- رفض الولد سماع النصيحة .
- قدم الزوج هدية الى زوجته .
- عرض التلفاز أفلاماً عربية و أجنبية .
- جلس الأولاد في غرفتهم .
- سرق اللصوص السيارة .
- صعد السكان بالمصعد .
- رجعت الأم من السوق .
- سبّح المصطافون في البحر .
- شرب الطفل اللبن .
- درس جون اللغة العربية .
- كسر الطفل الكوب .
- من حفر حفرة لأخيه وقع فيها .
- نظر المدير الى التقرير .
- خرج المشاهدون من المسرح .
- طبخت الأم الطعام .
- سكن الأطفال مع جدتهم .
- غسل الطفل يده بالماء والصابون .

ثانياً : الفعل المعتل :

تعريفه : وهو ما كان في حروفه الأصلية حرف أو اثنين من حروف العلة و هي : الألف و الواو و الياء .  
مثل : دعا ، باع ، طوى ، رمى ، وفى ، صام ، وثب .  
والفعل يصبح معتلاً إذا اعتلت فاؤه مثل : وعد ، يسر ، أو عينه مثل : قام ، باع ، أو لامه مثل : دعا ، رمى ، أو فاؤه و لامه مثل : وفى ، وفى ، أو عينه ولامه  
مثل : طوى ، نوى .

أقسام الفعل المعتل :

• المثال :

- تعريفه: وهو ما كان أول حروفه الأصلية حرف علة .  
مثل : وعد ، ينس ، وجد ، وهب ، وصل ، وضع ، وثق ، وقف ، وفد ، وجه ، وعظ ، ودع ، وفر ، وقع ، وزع . ولد ، وصف ، وزن ، وضع ، وضع .
- وعد الشاب والديه بالجتهاد فى الدراسة .
  - ينس الطالب الكسول من النجاح .
  - وجد الطفل الحلوى .
  - وهب الله العالم الذكاء .
  - وصل المسافر الى وطنه .
  - وضع الأستاذ الخريطة على الحائط .

• وثق الأب في أخلاق ولده.

• الأجوف :

تعريفه: و هو ما كان ثاني حروفه الأصلية حرف علة .  
مثل : سار ، مال ، قال ، طاب، نال، زار، باع، صاح، نام، غاب، عاد، زاد، عاش ، جاع، خاض، فاز ، قام، طاف، طار، طال، بات، حاز، قاد، ثار، باع، خاف، خان، جاء.

- سار التلميذ الى المدرسة.
- مالت الشجرة المحملة بالثمار.
- قال الرجل الحقيقة .
- طاب الثمر على الشجر.
- نال الطالب المجتهد الجائزة.
- زار السائح مصر.
- باع التاجر بضاعته.
- صاح الديك فجراً .
- نام الشاب.
- غاب الطالب عن المدرسه لمرضه.
- عاد المسافرون الى أرض الوطن.
- زاد المصنع من أنتاجه.
- عاش الجد عمراً مديداً.
- جاع الرجل الفقير .
- خاض المصريون حرب أكتوبر.
- فاز كرم جابر بميدالية أوليمبية.
- قام الطالب بأداء واجباته.

• الناقص :

تعريفه: و هو ما كان آخر حروفه الأصلية حرف عله .  
مثل : بنى ، رضى ، دعا ، دنا ، رمى ، لقي، بكى، مضى، نسي، نهى، جرى، غنى، شفى، خشى، بقى، ولى، نجا، شكاً، بدا ، أبى، أتى.

- بنى المهندس منزلاً .
- رضى المعلم عن أجهتاد تلاميذه .
- دعا القائد الى السلام .
- دنا الطالب من المدرسة.
- رمى الطفل الورق في سلة المهملات.
- لقي الولد صديقه.
- بكى الطفل على فراق والديه.
- مضى القطار في طريقه.
- نسى المريض ميعاد الطبيب.
- نهى الأستاذ التلاميذ عن التحدث أثناء الدرس.
- جرى اللاعب وراء الكرة.
- غنى المطرب على المسرح.
- شفى الله المريض.
- خشى اللص من الشرطي.
- بقى الزوج بجوار زوجته المريضه.
- ولى اللص هارباً .
- نجا الرجل من الموت.
- شكوا المعلم من الضوضاء في الفصل.
- بدا الغضب على وجه الشاب.
- أبى المقاتلون الاستسلام.
- أتى الرجل الى المنزل.

Exercises:

أستخرج الفعل الصحيح:

This set of questions measures only one concept per learning item, in addition, it also measures students' ability to understand a single learning item which is el fe3l el sa7i7 and that's why it is considered of **Difficulty level one.**

- رضى وعد مال أكل
- وعظ سار بنى أخذ
- مر نال وهب رمى
- أمر دعا ينس قال
- دنا قرأ طاب وجد
- زار وصل لقي رد
- وضع باع مضى عد
- جرى وثق صاح دق
- فهم نسي وقف نام
- نهى لعب غاب وفد
- بكى عاد رسم نسي
- وقف زاد نقل ولى

This set of questions measures only one concept per learning item, however, because it is a multiple choice question with two possible right answers, it will be considered of **Difficulty level Two.**

أستخرج الفعلين الصحيحين:

- عمل وعد دنا جف
- زرع وجد هز قال
- عز سار وهب فتح
- مال شد ينس كتب
- وصل أمن مضى فهم
- رد سال وضع بكى
- بدأ وثق زار عد
- أخذ باع لعب لقي
- طاب رسم صاح قرأ
- نقل بنى أكل وقف
- رضى ضم نال سنم
- دق دعا أمر رمى

أستخرج الأفعال الصحيحة:

This set of questions measures only one concept per learning item, however, because it is a multiple choice question with two or more possible right answers, it will be considered of **Difficulty level three.**

- سنم قص قدم سال
- آخر رفض شد ينس
- ملا ضم لعب سال
- حل قرأ زرع بنى
- صب فهم كتب أمن
- سد رسم دنا رضى
- تم أمر زرع فتح
- هم نقل وهب طاب
- خرج أكل عز سار
- طبخ مر فتح وجد
- سكن رغب بدأ فتح

- غسل مَدَّ أخذ وعد

أستخرج الفعل المعتل:

This set of questions measures only one concept per learning item, in addition, it also measures student's ability to understand a single learning item which is el fe3l el mo3tal and that's why it is considered of **Difficulty level one.**

- وعد شدَّ زرع عرض
- ينس عزَّ عمل ظلُّ
- وجد فتح نقل مرَّ
- وهب جفَّ آخر نظر
- سار بدأ رفض ضمَّ
- مال قرأ رغب نشأ
- قال أخذ قصَّ سئم
- طاب كتب أمر مَدَّ
- بنى لعب دقَّ ملأ
- رضى هزَّ فهم أكل
- دعا سأل ردَّ قدم
- دنا أمن رسم عدَّ

أستخرج الفعلين المعتلين:

This set of questions measures only one concept per learning item, however, because it is a multiple choice question with two possible right answers, it will be considered of **Difficulty level Two.**

- تمَّ سئم وضع نجا
- ودع قاد حضر ملأ
- بات نشأ غسل جاء
- آخر أمر وفر حاز
- أبى درس ظنَّ أتى
- ثار أكل حلَّ وضع
- وقع طبخ باع قرأ
- خاف أخذ سدَّ خشى
- قال كسر قلَّ وزن
- كسر حفر بقى ولى
- همَّ خرج وصف شكَا
- خان سكن ولد صبَّ

أستخرج الأفعال المعتلة:

This set of questions measures only one concept per learning item, however, because it is a multiple choice question with two or more possible right answers, it will be considered of **Difficulty level three.**

- وجه طار خرج طبخ
- بقى أبى نام وعظ
- حاز نشأ ودع صاح
- جاء سئم وفر باع
- شفى ملأ ولى وقع
- ولد طال آخر بقى
- وصف بات غسل وصل
- خان همَّ سكن وزن
- ثار وضع بدا تمَّ

- قاد نجا ولى وضع
- باع غنى وجه وعظ
- خاف خشى أتى شكاً

True/False Exercises:

Topic Covered	Learn Item Covered	Correct Answer	Difficulty Level	الجملة
1	3	False	2	الفعل السالم هو أحد أقسام الفعل المعتل
1	1	False	2	الفعل المهموز هو أحد أقسام الفعل المعتل
1	2	False	2	الفعل المضعف الثلاثي هو أحد أقسام الفعل المعتل
2	6	True	2	الفعل الناقص هو أحد أقسام الفعل المعتل
2	4	True	2	الفعل المثال هو أحد أقسام الفعل المعتل
2	5	True	2	الفعل الأجوف هو أحد أقسام الفعل المعتل
2	6	False	2	الفعل الناقص هو أحد أقسام الفعل الصحيح
2	4	False	2	الفعل المثال هو أحد أقسام الفعل الصحيح
2	5	False	2	الفعل الأجوف هو أحد أقسام الفعل الصحيح
1	1	True	2	الفعل المهموز هو أحد أقسام الفعل الصحيح
1	3	True	2	الفعل السالم هو أحد أقسام الفعل الصحيح
1	2	True	2	الفعل المضعف الثلاثي هو أحد أقسام الفعل الصحيح
1	1,2,3	True	1	أقسام الفعل الصحيح هي: المهموز، السالم و المضعف الثلاثي
1	1,2,3	False	1	أقسام الفعل الصحيح هي: المهموز، السالم و الأجوف
1	1,2,3	False	1	أقسام الفعل الصحيح هي: المهموز، المثال و الأجوف
1	1,2,3	False	1	أقسام الفعل الصحيح هي: الناقص، السالم و الأجوف
2	4,5,6	True	1	أقسام الفعل المعتل هي: الناقص، المثال و الأجوف
2	4,5,6	False	1	أقسام الفعل المعتل هي: الناقص، المهموز و الأجوف
2	4,5,6	False	1	أقسام الفعل المعتل هي: الناقص، المثال و المضعف
2	4,5,6	False	1	أقسام الفعل المعتل هي: المهموز، المثال و الأجوف
2	4,5,6	False	1	أقسام الفعل المعتل هي: المهموز، السالم و المضعف الثلاثي
1	1,2,3	True	1	الفعل الصحيح هو ما خلت حروفه الأصلية من أحرف العلة (الواو، الألف، الياء)
2	4,5,6	False	1	الفعل المعتل هو ما خلت حروفه الأصلية من أحرف العلة (الواو، الألف، الياء)
2	4,5,6	True	1	أحرف العلة هي (الواو، الألف، الياء)
2	4,5,6	False	1	أحرف العلة هي (الواو، الألف، التاء)
2	4,5,6	False	1	أحرف العلة هي (الواو، الألف، الياء)
1	2	True	2	الفعل المضعف الثلاثي هو ما كان وسطه و آخره من جنس واحد
1	2	True	2	الفعل المضعف الثلاثي هو ما كان ثاني حروفه الأصلية وثالثها من جنس واحد
2	4	False	2	الفعل المثال هو ما كان وسطه و آخره من جنس واحد
1	3	True	2	الفعل السالم هو ما سلمت حروفه الأصلية من الهمز و التضعيف
2	4	False	2	الفعل المثال هو ما سلمت حروفه الأصلية من الهمز و التضعيف
1	1,2,3	False	1	الفعل الصحيح هو ما كان في حروفه الأصلية حرف أو اثنان من حروف العلة و هي : الألف و الواو و الياء



1	1,2,3	False	1	الفعل الصحيح هو ما اعتلت فازه
1	1,2,3	False	1	الفعل الصحيح هو ما اعتلت عينه
1	1,2,3	False	1	الفعل الصحيح هو ما اعتلت لامه
1	1,2,3	False	1	الفعل الصحيح هو ما اعتلت فازه و لامه
1	1,2,3	False	1	الفعل الصحيح هو ما اعتلت عينه و لامه
2	4,5,6	True	1	الفعل المعتل هو ما كان في حروفه الاصلية حرف أو اثنتان من حروف العلة و هي : الألف و الواو و الياء
2	4,5,6	True	1	الفعل المعتل هو ما اعتلت فازه
2	4,5,6	True	1	الفعل المعتل هو ما اعتلت عينه
2	4,5,6	True	1	الفعل المعتل هو ما اعتلت لامه
2	4,5,6	True	1	الفعل المعتل هو ما اعتلت فازه و لامه
2	4,5,6	True	1	الفعل المعتل هو ما اعتلت عينه و لامه
2	4	True	2	الفعل المثال هو ما كان أول حروفه الاصلية حرف علة
2	4	False	2	الفعل المثال هو ما كان ثاني حروفه الاصلية حرف علة
2	4	False	2	الفعل المثال هو ما كان آخر حروفه الاصلية حرف علة
2	5	False	2	الفعل الأجوف هو ما كان أول حروفه الاصلية حرف علة
2	5	True	2	الفعل الأجوف هو ما كان ثاني حروفه الاصلية حرف علة
2	5	False	2	الفعل الأجوف هو ما كان آخر حروفه الاصلية حرف علة
2	6	False	2	الفعل الناقص هو ما كان أول حروفه الاصلية حرف عله
2	6	False	2	الفعل الناقص هو ما كان ثاني حروفه الاصلية حرف عله
2	6	True	2	الفعل الناقص هو ما كان آخر حروفه الاصلية حرف عله

**Appendix D: Sample Source Code**

**Pattern 1: Whole Part**

**Whole Part First Occurrence**

**TeachingMaterial class**

```
public class TeachingMaterial
{private Vector topics;
.....}
```

**Topic class**

```
{private Vector includedLearningItems;
private Vector excercises;
.....}
```

**LearnItem class**

```
private Vector examples;
{.....}
```

**Exam class**

```
public class Exam implements Sortable
{private Vector studentExcercises;
private int id;
private double score;
private double fullMark;
private int percentage;
private String category="";
private Timestamp examDate;
private int currentExcerciseIndex=-1;
private boolean isEntryExam=false;
private boolean isRunningExam=false;
private int order=0;}
```

```
public class StudentExcercise
```

```
{private Excercise excercise;  
private HashMap learnItemsExamples;  
private Vector studentAnswers;  
private int grade;  
private int order=1;  
private int tries;  
private boolean isExcerciseSolved=false;  
private boolean isAnswerRight=false; }
```

**Pattern 2: Singleton**

**Singleton First Occurrence**

**TeachingMaterial Class**

```
private static TeachingMaterial material=null;
private TeachingMaterial()
{topics=new Vector();
this.loadTopics();}
public static TeachingMaterial getMaterial()
{if(material==null)
{material=new TeachingMaterial();}
return material;}
```

**Singleton Second Occurrence**

**ArabicAlphabeticBundle Class**

```
private static ArabicAlphabeticBundle bundle=null;
private ArabicAlphabeticBundle()
{rb=ResourceBundle.getBundle("arabicunicode");}
public static ArabicAlphabeticBundle getInstance()
{if(bundle==null)
{bundle=new ArabicAlphabeticBundle();
load();}
return bundle;}
```

**Singleton Third Occurrence**

**CollectiveStudentsInfo**

```
private Vector students;
private static CollectiveStudentsInfo studentInfo=null;
private CollectiveStudentsInfo()
{this.students=new Vector();
this.loadStudents();}
```

```
public static CollectiveStudentsInfo getCollectiveStudentInfoInstance()
{if(studentInfo==null)
{studentInfo=new CollectiveStudentsInfo();}
return studentInfo;}
```

### Singleton Fourth Occurrence

#### **UserInterfaceBundle Class**

```
private static UserInterfaceBundle arabicBundle=null;
private static UserInterfaceBundle englishBundle=null;
private UserInterfaceBundle(Locale locale)
{rb=ResourceBundle.getBundle("userInterface",locale);
load();}
public static UserInterfaceBundle getArabicInstance()
{if(arabicBundle==null)
{arabicBundle=new UserInterfaceBundle(new Locale("ar","eg"));}
return arabicBundle;}
public static UserInterfaceBundle getEnglishInstance()
{if(englishBundle==null)
{englishBundle=new UserInterfaceBundle(new Locale("en","us"));}
return englishBundle;}
```

### Pattern 3: New Flyweight

#### New Flyweight First Occurrence

##### ArabicAlphabeticBundle class

```
private static Letter[] allEntries;

public Letter getLetterOfDecimalCode(int code)
{String hexaString=Integer.toHexString(code).toUpperCase();
for(int i=0;i<allEntries.size();i++)
{Letter current=(Letter)allEntries.elementAt(i);
if(current.getUnicode().equals(hexaString))
{return current;}}
return null;}
```

##### Letter class

```
private String unicode;
private String englishRepresentation;
```

#### New Flyweight Second Occurrence

##### TeachingMaterial class

```
private Topic[] topics;

public Topic getTopicById(int topicId)
{for(int i=0;i<topics.size();i++)
{Topic currentTopic=(Topic)topics.elementAt(i);
if(currentTopic.getId()==topicId)
return currentTopic;}
return null;}
```

##### Topic class

```
private Exercise[] exercises;

public Exercise getExerciseById(int exerciseId)
{for(int i=0;i<this.exercises.size();i++)
{Exercise currentExercise=(Exercise)this.exercises.elementAt(i);
```

```
if(currentExercise.getId()==exerciseId)
```

```
return currentExercise;}
```

```
return null;}
```

```
LearnItem class :
```

```
private Example[] examples;
```

```
public Example getExampleById(int exampleId)
```

```
{for(int i=0;i<examples.size();i++)
```

```
{Example currentExample=(Example)examples.elementAt(i);
```

```
if(currentExample.getId()==exampleId)
```

```
return currentExample;}
```

```
return null;}
```

## Pattern 4: Builder

## TeachingMaterial Class

```

public class TeachingMaterial
{
    private void loadTopics()
    {
        ITSDatabase dbBean=new ITSDatabase();
        dbBean.executeQuery("select * from topics");
        while(dbBean.next())
        {
            int topicId=dbBean.getInt("TOPIC_ID");
            String topicTitle=dbBean.getString("title");
            int topicSequence=dbBean.getInt("sequence");
            String statement=dbBean.getClob("statement");
            Topic topic=new Topic(topicTitle);
            topic.setId(topicId);
            topic.setSequence(topicSequence);
            topic.setStatement(statement);
            topics.add(topic);}
        dbBean.commit();
        dbBean.closeAll();
        for(int i=0;i<topics.size();i++)
        {
            Topic currentTopic=(Topic)topics.elementAt(i);
            currentTopic.load();
            LearnItem[] items=currentTopic.getAllIncludedLearningItems();
            for(int j=0;j<items.length;j++)
            {
                items[j].loadExamples();}
            this.sortTopics();}
    }
}

public class Topic implements Sortable
{
    public void load()
    {
        this.loadLearningItems();
    }
}

```



```

this.loadExercices();}

private void loadLearningItems()
{ITSDatabase dbBean=new ITSDatabase();
dbBean.executeQuery("select * from learning_item where topic_id="+this.getId());
while(dbBean.next())
{LearnItem item=new LearnItem(dbBean.getString("title"));
item.setId(dbBean.getInt("learning_item_id"));
item.setSequence(dbBean.getInt("sequence"));
item.setStatement(dbBean.getClob("statement"));
item.load();
this.includedLearningItems.addElement(item);}
dbBean.closeAll();
this.sortLearningItems();}}

public class Topic implements Sortable
{public void load()
{ITSDatabase dbBean=new ITSDatabase();
dbBean.executeQuery("select * from examples where
learning_item_id="+super.getId());
while(dbBean.next())
{int exampleId=dbBean.getInt("example_id");
String keyWord=dbBean.getString("keyword");
String statement=dbBean.getClob("statement");
int sequence=dbBean.getInt("sequence");
Example example=new Example(exampleId,keyWord,statement,sequence);
this.examples.addElement(example);}
dbBean.closeAll();}}

```

**Pattern 5: New Adapter****New Adapter First Occurrence****Exercise class**

```
public class Exercise
{
    private String questionText;
    private int difficultyLevel;
    private int id;
    private String type;
    private Vector coveredItems;

    public Exercise (String questionText,int difficultyLevel)
    {this.questionText=Utilities.convertString(questionText);
    this.difficultyLevel=difficultyLevel;
    this.coveredItems=new Vector();}

    public int getId()
    {return id;}

    public void setId(int id)
    {this.id = id;}

    public int getDifficultyLevel()
    {return difficultyLevel;}

    public void setDifficultyLevel(int difficultyLevel)
    {this.difficultyLevel = difficultyLevel;}

    public String getQuestionText()
    {return questionText;}

    public void setQuestionText(String questionText)
    {this.questionText = Utilities.convertString(questionText);}

    public String getType()
    {return type;}

    public void setType(String type)
```

```

{this.type = type;}
public void addCoveredItem(int itemId)
{if(!this.coveredItems.contains(itemId+""))
    this.coveredItems.addElement(itemId+"");}
public int[] getAllCoveredItems()
{int[] items=new int[this.coveredItems.size()];
for(int i=0;i<this.coveredItems.size();i++)
{items[i]=Integer.parseInt(this.coveredItems.elementAt(i).toString())}
return items;}
public void clearAllCoveredItems()
{this.coveredItems=new Vector();}
public boolean isLearnItemCovered(int itemId)
{for(int i=0;i<this.coveredItems.size();i++)
{if(Integer.parseInt(this.coveredItems.elementAt(i).toString())==itemId)
    return true;}
return false;}
public String getTypeString()
{return this.type;}}

```

### **StudentExcercise class**

```

public class StudentExcercise extends Excercise
{private Example[] learnItemsExamples;
private String[] studentAnswers;
private int grade;
private int order=1;
private int tries;
private boolean isExcerciseSolved=false;
public StudentExcercise()
{ studentAnswers=new Vector();

```

```
learnItemsExamples=new HashMap();
tries=0;}
public Exercise getExercise()
{return exercise;}
public void setExercise(Exercise exercise)
{this.exercise = exercise;}
public int getGrade()
{return grade;}
public void setGrade(int grade)
{this.grade = grade;}
public int getOrder()
{return order;}
public void setOrder(int order)
{this.order = order;}
public void addMCQAnswer(PossibleAnswer answer)
{this.studentAnswers.addElement(answer);}
public void addYesNoAnswer(boolean answer)
{this.studentAnswers.addElement(new Boolean(answer));}
public double getScore()
{if(this.isAnswerRight())
{ if(this.tries==1)
return this.getExercise().getDifficultyLevel()*2;
else
return this.getExercise().getDifficultyLevel();}
else
return 0.0;}
public boolean isAnswerRight()
{boolean isAnswerRight=true;
```

```

if(this.exercice instanceof MCQExercice)
{MCQExercice e=(MCQExercice)exercice;
int numberOfRightAnswers=e.getNumberOfRightAnswers();
if(studentAnswers.size()!=numberOfRightAnswers)
{ isAnswerRight=false;}
else
{for(int i=0;i<studentAnswers.size();i++)
{ if(!((PossibleAnswer)studentAnswers.elementAt(i)).isRightAnswer())
{isAnswerRight=false;
break;}}}}
else if (this.exercice instanceof YesNoExercice)
{YesNoExercice e=(YesNoExercice)exercice;
boolean studentAnswer=((Boolean)this.studentAnswers.elementAt(0)).booleanValue();
if((e.isRight() && studentAnswer) || (!e.isRight() && !studentAnswer))
isAnswerRight=true;
else
isAnswerRight=false;}
return isAnswerRight;}

public double getFullMark()
{return this.exercice.getDifficultyLevel()*2;}

public void clearStudentAnswers()
{this.studentAnswers=new Vector();}

public int getNumberOfTries()
{return tries;}

public void addTry()
{this.tries++;}

public int getTries()
{return tries;}

```

```

public void setTries(int tries)
{this.tries = tries;}

public void addLearnItemExamples(int learnItemId, Example[] examples)
{this.learnItemsExamples.put(learnItemId+ "", examples);}

public Example[] getLearnItemExamples(int learnItemId)
{return (Example[])this.learnItemsExamples.get(learnItemId+ "");}

public void setExcerciseAsSolved()
{this.isExcerciseSolved=true;}

public boolean isExcerciseSolved()
{return this.isExcerciseSolved;}}

```

**New Adapter Second Occurrence**

**Topic class**

```

package mypackage;

import java.util.*;

public class Topic implements Sortable
{private LearnItem[] includedLearningItems;

private Excercise[] excercises;

public Topic(String title)
{super(title);

includedLearningItems=new Vector();

excercises=new Vector();}

public int getLearnItemIndex(LearnItem item)
{return this.includedLearningItems.indexOf(item);}

public LearnItem getLearnItemByIndex(int index)
{return (LearnItem)this.includedLearningItems.elementAt(index);}

public int getNumberOfLearningItems()
{return this.includedLearningItems.size();}

private void sortLearningItems()

```

```

{this.includedLearningItems=Utilities.sortObjects(this.includedLearningItems);}
public Excercise[] getAllExcercises()
{Excercise[] allExcercises=new Excercise[excercises.size()];
this.excercises.toArray(allExcercises);
return allExcercises;}
public void loadExcercises()
{ITSDaTaBase db1=new ITSDaTaBase();
db1.executeQuery("select * from all_topic_excercises where topic_id="+this.getId());
while(db1.next())
{int excerciseId=db1.getInt("excercise_id");
Excercise excercise=this.getExcerciseById(excerciseId);
if(excercise!=null)
{excercise.addCoveredItem(db1.getInt("learning_item_id"));
continue;}
String type=db1.getString("type");
String questionText=db1.getString("question_text");
int difficultyLevel=db1.getInt("difficulty_level");
if(Defines.isMCQExcercise(type))
{excercise=new MCQExcercise(questionText,difficultyLevel);
MCQExcercise e=(MCQExcercise) excercise;
ITSDaTaBase db2=new ITSDaTaBase();
db2.executeQuery("select * from excercise_possible_answers where
excercise_id="+excerciseId);
while(db2.next())
{String answerText=db2.getString("possible_answer");
int answerId=db2.getInt("possible_answer_id");
PossibleAnswer answer=null;
if(db2.getString("is_right_answer").equals(Defines.POSSIBLE_ANSWER_IS_RIGHT))

```

```

answer=new PossibleAnswer(answerText,true);

else
    answer=new PossibleAnswer(answerText,false);
answer.setOrder(answerId);
e.addPossibleAnswer(answer);}
db2.closeAll(); }

else if(type.equals(Defines.EXERCISE_TYPE_ENTER_VERB))
{exercise=new Exercise(questionText,difficultyLevel);}
else if(type.equals(Defines.EXERCISE_TYPE_YES))
{exercise=new YesNoExercise(questionText,difficultyLevel,true);}
else if(type.equals(Defines.EXERCISE_TYPE_NO))
{exercise=new YesNoExercise(questionText,difficultyLevel,false);}

exercise.setId(exerciseId);

exercise.setType(type);

exercise.addCoveredItem(db1.getInt("learning_item_id"));

this.exercises.addElement(exercise);}

db1.closeAll();}

public void updateExistingExercise(int exerciseId,String type,String questionText,int
difficultyLevel,Vector coveredItems)
{ITSDatabase dbBean=new ITSDatabase();

Exercise e=this.getExerciseById(exerciseId);

e.setDifficultyLevel(difficultyLevel);

e.setQuestionText(questionText);

if(Defines.isYesNoExercise(type))
{e.setType(type);

YesNoExercise exercise=(YesNoExercise)e;

if(type.equals(Defines.EXERCISE_TYPE_YES))

exercise.setRightness(true);

```



```

else
    excercise.setRightness(false);}
dbBean.executeUpdate("update excercises set
question_text='"+Utilities.convertString(questionText)+"',difficulty_level='"+difficultyLevel+"',typ
e='"+type+"' where excercise_id='"+excerciseId);
dbBean.commit();
dbBean.executeUpdate("delete from excercise_learning_items where
excercise_id='"+excerciseId);
dbBean.commit();
e.clearAllCoveredItems();
for(int i=0;i<coveredItems.size();i++)
{dbBean.executeUpdate("insert into excercise_learning_items (excercise_id,learning_item_id)
values ('"+excerciseId+"','"+coveredItems.elementAt(i).toString()+")");
e.addCoveredItem(Integer.parseInt(coveredItems.elementAt(i).toString()));
dbBean.commit();}
dbBean.closeALI();}
public void updateExistingMCQExcercise(int excerciseId,String type,String questionText,int
difficultyLevel,Vector coveredItems,Vector answers)
{this.updateExistingExcercise(excerciseId,type,questionText,difficultyLevel,coveredItems);
MCQExcercise e=(MCQExcercise)this.getExcerciseById(excerciseId);
e.clearAnswers();
ITSDataBase dbBean=new ITSDataBase();
for(int i=0;i<answers.size();i++)
{ PossibleAnswer answer=(PossibleAnswer)answers.elementAt(i);
e.addPossibleAnswer(answer);
String isRightAnswer=Defines.POSSIBLE_ANSWER_IS_WRONG;
if(answer.isRightAnswer())
{isRightAnswer=Defines.POSSIBLE_ANSWER_IS_RIGHT;}}

```

```

String sqlStatement="insert into excercise_Possible_Answers
(exercise_id,possible_answer_id,possible_answer,is_right_answer)
values("+e.getId()+","+ (i+1) +","+Utilities.convertString(answer.getPossibleAnswer())+""," +isR
ightAnswer+"");
    dbBean.executeUpdate(sqlStatement);
    dbBean.commit();}
    dbBean.closeAll();}
public void addNewExcercise(Excercise newExcercise)
{this.excercises.addElement(newExcercise);
    ITSDatabase dbBean=new ITSDatabase();
    dbBean.executeQuery("select * from excercises");
    int maximumId=0;
    while(dbBean.next())
    {int exerciseId=dbBean.getInt("exercise_ID");
        if(exerciseId> maximumId)
            maximumId=exerciseId;}
    int nextId= maximumId+1;
    newExcercise.setId(nextId);
    dbBean.executeUpdate("insert into excercises
(exercise_id,difficulty_level,type,question_text)
values("+nextId+" ,"+newExcercise.getDifficultyLevel()+"," +newExcercise.getType()+"," +Utilit
ies.convertString(newExcercise.getQuestionText()+""));
    dbBean.commit();
    int[] coveredItems=newExcercise.getAllCoveredItems();
    for(int i=0;i<coveredItems.length;i++)
    {dbBean.executeUpdate("insert into excercise_learning_items (exercise_id,learning_item_id)
values (" +newExcercise.getId()+"," +coveredItems[i]+""));
        dbBean.commit();}

```

```

dbBean.closeAll();}

public Exercise getExerciseById(int exerciseId)
{for(int i=0;i<this.exercises.size();i++)
{Exercise currentExercise=(Exercise)this.exercises.elementAt(i);
if(currentExercise.getId()==exerciseId)
return currentExercise;}

return null;}

public void deleteExercise(int exerciseId)
{this.exercises.remove(this.getExerciseById(exerciseId));
ITSDatabase dbBean=new ITSDatabase();
String sqlStatement="delete from exercises where exercise_id="+exerciseId;
dbBean.executeUpdate(sqlStatement );
dbBean.commit();
dbBean.closeAll();}

public void addNewMCQExercise(MCQExercise newExercise)
{this.addNewExercise(newExercise);
ITSDatabase dbBean=new ITSDatabase();
PossibleAnswer[] possibleAnswers=newExercise.getPossibleAnswers();
for(int i=0;i<possibleAnswers.length;i++)
{String isRightAnswer="N";
if(possibleAnswers[i].isRightAnswer())
{isRightAnswer="Y";}
String sqlStatement="insert into exercise_Possible_Answers
(exercise_id,possible_answer_id,possible_answer,is_right_answer)
values("+newExercise.getId()+","+ (i+1)+","+Utilities.convertString(possibleAnswers[i].getPos
sibleAnswer()+","+isRightAnswer+"");
dbBean.executeUpdate(sqlStatement);
dbBean.commit();}

```

```

dbBean.closeAll();}

public Vector getExercicesWithDifficultyLevel(int level)
{Vector e=new Vector();
for(int i=0;i<this.exercices.size();i++)
{Exercice exercice=(Exercice)this.exercices.elementAt(i);
if(exercice.getDifficultyLevel()==level)
{e.addElement(exercice);}}
return e;}

public Vector getRandomExercices(int number)
{Vector randomExercices=new Vector();
for(int i=1;i<=3;i++)
{Vector e=this.getExercicesWithDifficultyLevel(i);
for(int j=0;j<number && j<e.size();j++)
{int exerciceIndex=(int)(Math.random()*(e.size()-1));
Exercice exercice=(Exercice)e.elementAt(exerciceIndex);
StudentExercice se=new StudentExercice();
se.setGrade(0);
se.setExercice(exercice);
randomExercices.addElement(se);
e.removeElement(exercice);}}
return randomExercices;}

public void addLearningItem(LearnItem item)
{ if(!includedLearningItems.contains(item))
{if(item.getSequence()>=((LearnItem)this.includedLearningItems.elementAt(this.includedLearnin
gItems.size()-1)).getSequence())
this.includedLearningItems.addElement(item);
else
{ for(int i=0;i<this.includedLearningItems.size();i++)

```

```

    if(((LearnItem)this.includedLearningItems.elementAt(i)).getSequence()>item.getSequence())
    {this.includedLearningItems.insertElementAt(item,i);
    return;}}}}

```

```

public LearnItem[] getAllIncludedLearningItems()
{LearnItem[] items=new LearnItem[this.includedLearningItems.size()];
this.includedLearningItems.toArray(items);
return items;}

```

```

public void loadLearningItems()
{ITSDatabase dbBean=new ITSDatabase();
dbBean.executeQuery("select * from learning_item where topic_id="+this.getId());
while(dbBean.next())
{LearnItem item=new LearnItem(dbBean.getString("title"));
item.setId(dbBean.getInt("learning_item_id"));
item.setSequence(dbBean.getInt("sequence"));
item.setStatement(dbBean.getString("statement"));
this.includedLearningItems.addElement(item);}
dbBean.closeAll();
this.sortLearningItems();}

```

```

public LearnItem getLearningItemById(int learningItemId)
{for(int i=0;i<this.includedLearningItems.size();i++)
{LearnItem currentItem=(LearnItem)this.includedLearningItems.elementAt(i);
if(currentItem.getId()==learningItemId)
{return currentItem;}}
return null;}

```

```

public void addNewItem(String title,int sequence,String statement)
{ITSDatabase dbBean=new ITSDatabase();
dbBean.executeQuery("select * from learning_item");
int maximumId=0;

```

```

while(dbBean.next())
{int itemId=dbBean.getInt("learning_item_ID");
  if(itemId>maximumId)
    maximumId=itemId;}
int nextId=maximumId+1;
LearnItem item=new LearnItem(title);
item.setId(nextId);
item.setSequence(sequence);
addLearningItem(item);
dbBean.executeUpdate("insert into learning_item
(learning_item_id,title,sequence,topic_id,statement)
values("+nextId+", "+Utilities.convertString(title)+" , "+sequence+" , "+super.getId()+" , "+Utiliti
es.convertString(statement)+"")");
  dbBean.closeAll();}

public LearnItem getLearnItemById(int itemId)
{for(int i=0;i<includedLearningItems.size();i++)
  {LearnItem currentItem=(LearnItem)includedLearningItems.elementAt(i);
    if(currentItem.getId()==itemId)
      return currentItem;}.
return null;}}

```

### StudentTopic class

```

package mypackage;
import java.util.*;
public class StudentTopic extends Topic implements Sortable
{private int viewCounter=0;
  private int currentLearnItemIndex=-1;
  private boolean isTopicStatementDisplayed=false;
  private String currentCategory;

```

```

public StudentTopic(Topic topic)
{this.topic=topic;
if(this.topic.getNumberofLearningItems(>0)
this.currentLearnItem=this.topic.getLearnItemByIndex(0);
exams=new Vector();
this.learnItemsExamples=new HashMap();}
public void loadExams(int studentId)
{ITSDatabase db1=new ITSDatabase();
db1.executeQuery("select * from exams where student_ID="+studentId+" and
topic_id="+this.topic.getId()+" and lower(is_entry_exam)='n'");
while(db1.next())
{int examId=db1.getInt("exam_ID");
Exam exam=new Exam();
ITSDatabase db2=new ITSDatabase();
db2.executeQuery("select * from exam_exercise where exam_id="+examId);
while(db2.next())
{int exerciseId=db2.getInt("exercise_id");
StudentExercise se=new StudentExercise();
se.setExercise(this.topic.getExerciseById(exerciseId));
se.setGrade(db2.getInt("student_grade"));
se.setTries(db2.getInt("tries"));
exam.addStudentExercise(se);; }
exam.setCategory(db1.getString("category"));
exam.setExamDate(db1.getDate("exam_date"));
exam.setExamTime(db1.getTime("exam_date"));
exam.setId(examId);
this.exams.addElement(exam);
db2.closeALI();}

```

```
db1.closeAll();  
this.sortExams();}  
private void sortExams()  
{this.exams=Utilities.sortObjects(this.exams);  
this.currentCategory=((Exam)this.exams.elementAt(this.exams.size()-1)).getCategory();}  
public int getViewCounter()  
{return viewCounter;}  
public void setViewCounter(int viewCounter)  
{this.viewCounter = viewCounter;}  
public Topic getTopic()  
{return topic;}  
public Exam getLastExam()  
{return (Exam)this.exams.elementAt(exams.size()-1);}  
public boolean moveToNextLearnItem()  
{if(this.currentLearnItemIndex>-1 &&  
this.currentLearnItemIndex<this.getTopic().getNumberOfLearningItems()-1)  
{this.currentLearnItemIndex++;  
this.currentLearnItem=this.getTopic().getLearnItemByIndex(this.currentLearnItemIndex);  
return true;}  
else  
return false;}  
public boolean moveToPreviousLearnItem()  
{if(this.currentLearnItemIndex>0)  
{this.currentLearnItemIndex--;  
this.currentLearnItem=this.getTopic().getLearnItemByIndex(this.currentLearnItemIndex);  
return true;}  
else  
return false;}
```



```

public boolean moveToFirstLearnItem()
{if(this.getTopic().getNumberOfLearningItems()>0)
{this.currentLearnItem=this.getTopic().getLearnItemByIndex(0);
this.currentLearnItemIndex=0;
this.isTopicStatementDisplayed=false;
return true;}
else
{return false;}}
public boolean moveToLastLearnItem()
{if(this.getTopic().getNumberOfLearningItems()>0)
{this.currentLearnItemIndex=this.getTopic().getNumberOfLearningItems()-1;
this.currentLearnItem=this.getTopic().getLearnItemByIndex(this.currentLearnItemIndex);
return true;}
else
{return false;}}
public LearnItem getCurrentLearnItem()
{return this.currentLearnItem;}
public boolean isFirstLearnItem()
{if(this.topic.getLearnItemIndex(this.currentLearnItem)==0)
return true;
else
return false;}
public boolean isLastLearnItem()
{if(this.topic.getLearnItemIndex(this.currentLearnItem)==this.topic.getNumberOfLearningItems
()-1)
return true;
else
return false;}

```

```

public boolean isTopicStatementDisplayed()
{return isTopicStatementDisplayed;}

public void setTopicStatementDisplayed(boolean isTopicStatementDisplayed)
{this.isTopicStatementDisplayed = isTopicStatementDisplayed;}

public void populateExamples(Student student)
{this.learnItemsExamples=new HashMap();

String category=this.getLastExam().getCategory();

int number=Defines.getNumberOfCategoryLearnItemExamples(category);

for(int i=0;i<this.getTopic().getNumberOfLearningItems();i++)
{LearnItem currentItem=this.getTopic().getLearnItemByIndex(i);

Example[]
notSeenExamples=currentItem.getExamplesNotSeen(number,student.getAllSeenExamples());

this.learnItemsExamples.put(""+currentItem.getId(),notSeenExamples);}}

public Example[] getExamples()
{return (Example[])this.learnItemsExamples.get(this.currentLearnItem.getId()+"");}

public void saveExamples(Student student)
{Iterator examplesIterator=this.learnItemsExamples.values().iterator();

while(examplesIterator.hasNext())
{Example[] examples=(Example[])examplesIterator.next();

for(int i=0;i<examples.length;i++)
{student.addStudentExample(examples[i]);}}}

public void addExam(Exam exam)
{this.exams.addElement(exam);}

public void generateExam(String category,Student student)
{int[] levels=Defines.getCategoryDifficultyLevelsExercices(category);

Exam exam=new Exam();

Vector allExercices=Utilities.convertToVector(this.getTopic().getAllExercices());

for(int i=0;i<this.exams.size();i++)

```

```

{Exam currentExam=(Exam)exams.elementAt(i);
 StudentExercise[] takenExercises=currentExam.getAllExercises();
 for(int j=0;j<takenExercises.length;j++)
 {allExercises.remove(takenExercises[j].getExercise());}}
for(int i=0;i<3;i++)
{Vector levelExercises=new Vector();
 for(int j=0;j<allExercises.size();j++)
 {Exercise currentExercise=(Exercise)allExercises.elementAt(j);
  if(currentExercise.getDifficultyLevel()==(i+1))
  {levelExercises.addElement(currentExercise);}}
for(int j=0;j<levels[i] && levelExercises.size()>0;j++)
 {int exerciseIndex=(int)(Math.random()*(levelExercises.size()-1));
  Exercise exercise=(Exercise)levelExercises.elementAt(exerciseIndex);
  StudentExercise se=new StudentExercise();
  se.setGrade(0);
  se.setExercise(exercise);
  int[] coveredItems=exercise.getAllCoveredItems();
  for(int k=0;k<coveredItems.length;k++)
  {LearnItem item=this.getTopic().getLearningItemById(coveredItems[k]);
   int number=Defines.getNumberOfCategoryExerciseExamples(category);
   Example[]
learnItemExamples=item.getExamplesNotSeen(number,student.getAllSeenExamples());
   se.addLearnItemExamples(item.getId(),learnItemExamples);}
  exam.addStudentExercise(se);
  levelExercises.removeElement(exercise);}}
exam.setExamDate(new Date());
this.currentExam=exam;
this.currentCategory=category;}
    
```

```

public Exam getCurrentExam()
{return currentExam;}

public void saveCurrentExam(int studentId)
{this.currentExam.saveExam(this.topic.getId(),studentId);
this.exams.addElement(this.currentExam);
this.currentExam=null;}

public String getCurrentCategory()
{return currentCategory;}

public Exam[] getAllExams()
{Exam[] allExams=new Exam[this.exams.size()];
this.exams.toArray(allExams);
return allExams;}

public Exam getEntryExam()
{for(int i=0;i<this.exams.size();i++)
{Exam exam=(Exam)this.exams.elementAt(i);
if(exam.isEntryExam())
return exam;}
return null;}

public boolean isAfter(Object otherObject)
{StudentTopic otherTopic=(StudentTopic)otherObject;
if( (this.getTopic().getSequence())>otherTopic.getTopic().getSequence())
||( otherTopic.getTopic().getSequence()==this.getTopic().getSequence()
&& Defines.getCategoryOrder(otherTopic.getLastExam().getCategory())
<Defines.getCategoryOrder(this.getLastExam().getCategory()))
return true;
else
return false;}}

```

**Pattern 6: New Strategy**

**Sortable interface**

```
public interface Sortable
{boolean isAfter(Object otherObject);}
```

**Strategy Method (sortObjects)**

```
public static void sortObjects(Sortable[] unsortedObjects)
{Sortable temp;
for(int i=0;i<unsortedObjects.length-1;i++)
for(int j=i+1;j<unsortedObjects.length;j++)
{if(unsortedObjects[i].isAfter(unsortedObjects[j]))
{temp=unsortedObjects[i];
unsortedObjects[i]=unsortedObjects[j];
unsortedObjects[j]=temp;}}}
```

**GeneralItem isAfter method**

```
public boolean isAfter(Object otherObject)
{GeneralItem otherItem=(GeneralItem)otherObject;
if(this.getSequence()>otherItem.getSequence())
return true;
else
return false;}
```

**Exam isAfter method**

```
public boolean isAfter(Object otherObject)
{Exam otherExam=(Exam)otherObject;
if(this.getExamDate().after(otherExam.getExamDate()))
return true;
else
return false;}
```

**StudentTopic isAfter method**

```

public boolean isAfter(Object otherObject)
{StudentTopic otherTopic=(StudentTopic)otherObject;
  if( (this.getTopic().getSequence()>otherTopic.getTopic().getSequence())
    ||( otherTopic.getTopic().getSequence()==this.getTopic().getSequence()
      && Defines.getCategoryOrder(otherTopic.getLastExam().getCategory())
        <Defines.getCategoryOrder(this.getLastExam().getCategory()))))
    return true;
  else
    return false;}

```

#### Example isAfter method

```

public boolean isAfter(Object otherObject)
{Example otherExample=(Example)otherObject;
  if(this.getSequence()>otherExample.getSequence())
    return true;
  else
    return false;}

```

**Pattern 7: Template Method**

**Exercise getTypeString method of class Exercise**

```
public String getTypeString()  
{return this.type;}
```

**YesNoExercise getTypeString method of class YesNoExercise**

```
public String getTypeString()  
{return "Yes/No";}
```

**MCQExercise getTypeString method of class MCQExercise**

```
public String getTypeString()  
{if(this.getType().equalsIgnoreCase(Defines.EXERCISE_TYPE_MCQ_ONE))  
    return "MCQ (One Right Answer)";  
else if(this.getType().equalsIgnoreCase(Defines.EXERCISE_TYPE_MCQ_TWO))  
    return "MCQ (Two Right Answers)";  
else if(this.getType().equalsIgnoreCase(Defines.EXERCISE_TYPE_MCQ_MANY))  
    return "MCQ (Choose All That Apply)";  
return "";}  
}
```

**Pattern 8: Master Slave**

**Exam class**

```
public class Exam
{public void calculateExam()
{score=0;
for(int i=0;i<this.studentExercises.size();i++)
```

**StudentExercise class**

```
currentExercise=(StudentExercise)this.studentExercises.elementAt(i);
score+=currentExercise.getScore();}
fullMark=0;
for(int i=0;i<this.studentExercises.size();i++)
{currentExercise=(StudentExercise)this.studentExercises.elementAt(i);
fullMark+=currentExercise.getFullMark();}
percentage= (int)Math.ceil(score*100.0/fullMark);
this.category=Defines.getCategoryOfPercentage(percentage);}
```

```
public class StudentExercise extends Exercise
{public double getScore()
{if(this.isAnswerRight())
{if(this.tries==1)
return this.getExercise().getDifficultyLevel()*2;
else
return this.getExercise().getDifficultyLevel();}
else return 0.0;}}
```



**Pattern 9: Observer**

**Observer interface**

```
public interface Observer
{void update(Object object);}
```

**Observable interface**

```
public interface Observable
{void registerObserver(Observer Observer);}
```

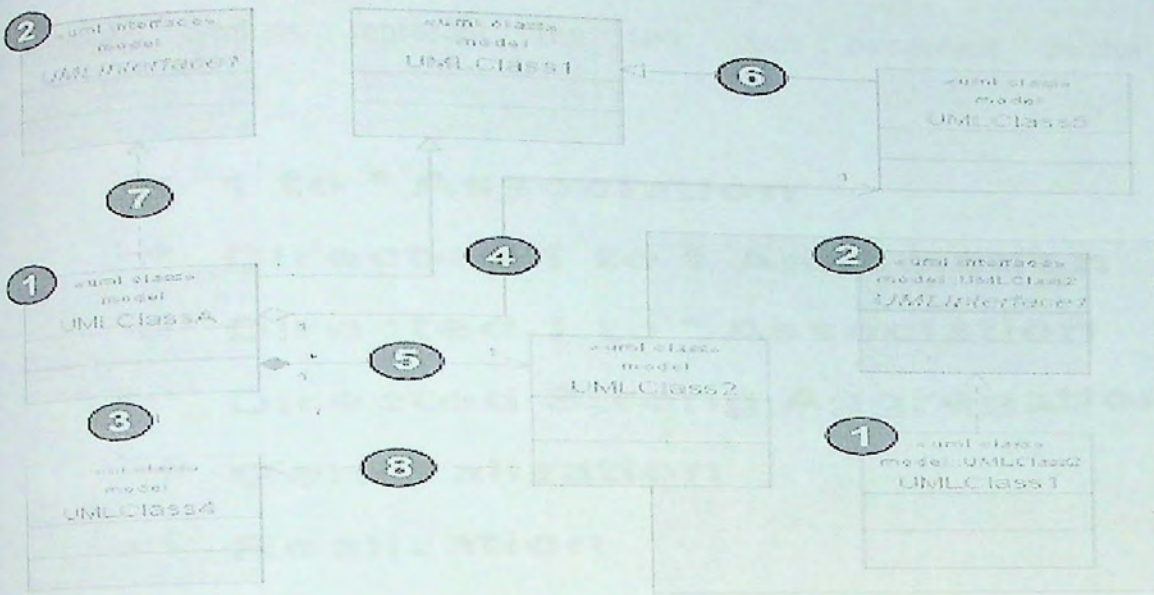
**Student class**

```
private Observer[] Observers;
public void registerObserver(Observer Observer)
{if(!this.Observers.contains(Observer))
{this.Observers.addElement(Observer);}}
public void updateObservers()
{for(int i=0;i<this.Observers.size();i++)
{((Observer)this.Observers.elementAt(i)).update(this);}}
```

**CollectiveStudentsInfo class**

```
public void update(Object object)
{if(object instanceof Student)
{Student student =(Student)object;
Student existingStudent=this.getStudentById(student.getId());
if(existingStudent==null)
{this.students.addElement(student);}
else
{this.students.remove(existingStudent);
this.students.addElement(student);}}}
```

## Appendix E: How to Read a UML Class Diagram



1. UML Class

2. UML Interface

3. Directed Association

4. Weakly Aggregated Association

5. Strongly Aggregated Association







6. Generalization Relationship

7. Realization Relationship

8. Dependency

### UML Class Component Palette:

The following symbols represent the UML class component palette in the Jdeveloper

-  **1 to \* Association**
-  **Directed 1 to 1 Association**
-  **Directed 1 to \* Association**
-  **Directed Strong Aggregation**
-  **Generalization**
-  **Realization**

#### Note:

In Java Class diagrams Generalization is sometimes referred to as extends and Realization is sometimes referred to as implements.

Appendix F: Detailed Class Diagrams

Pattern 1: Whole Part

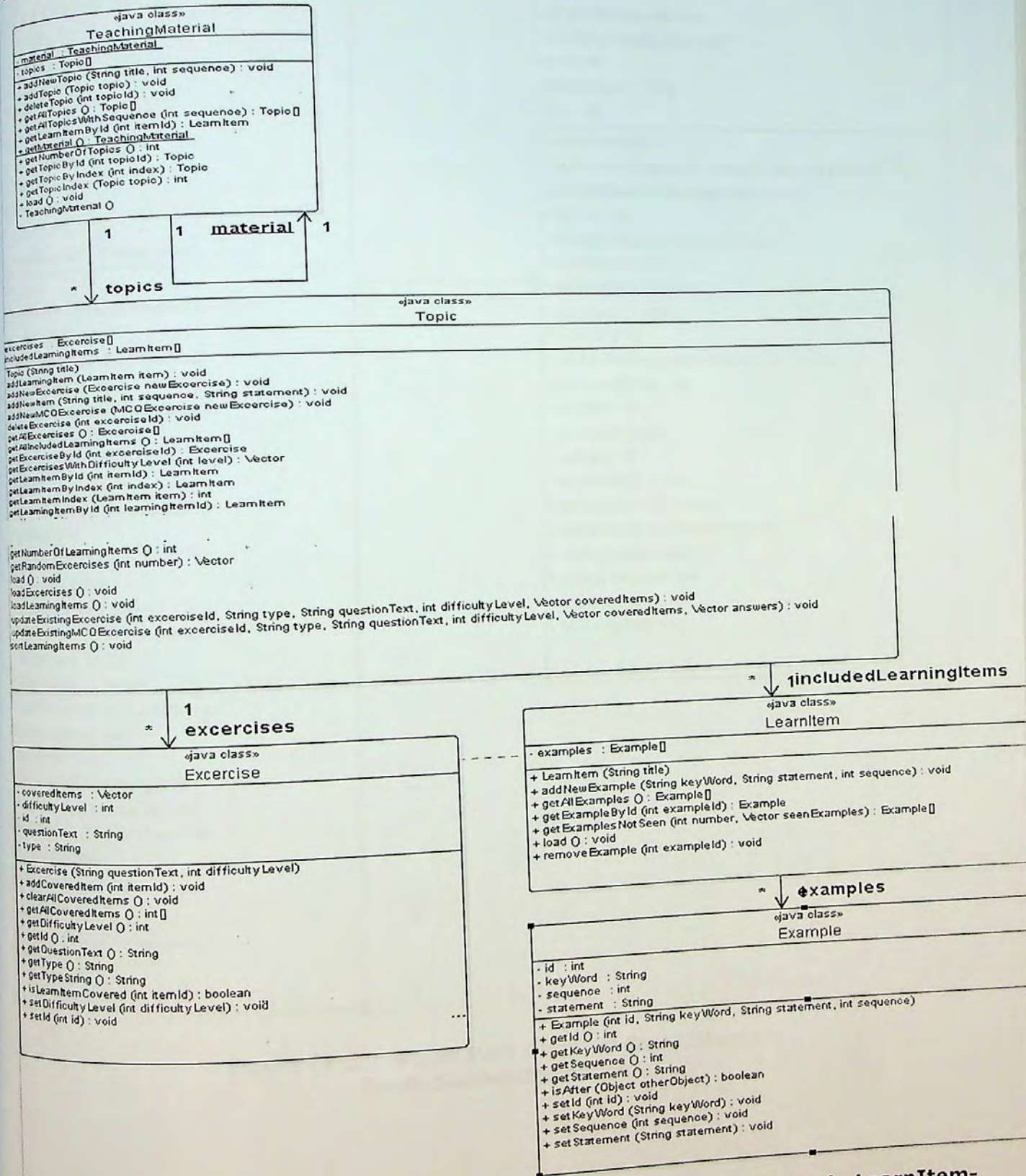


Figure (F-1): Whole Part Detailed Class Diagram. TeachingMaterial-Topic-LearnItem-Example-Exercise

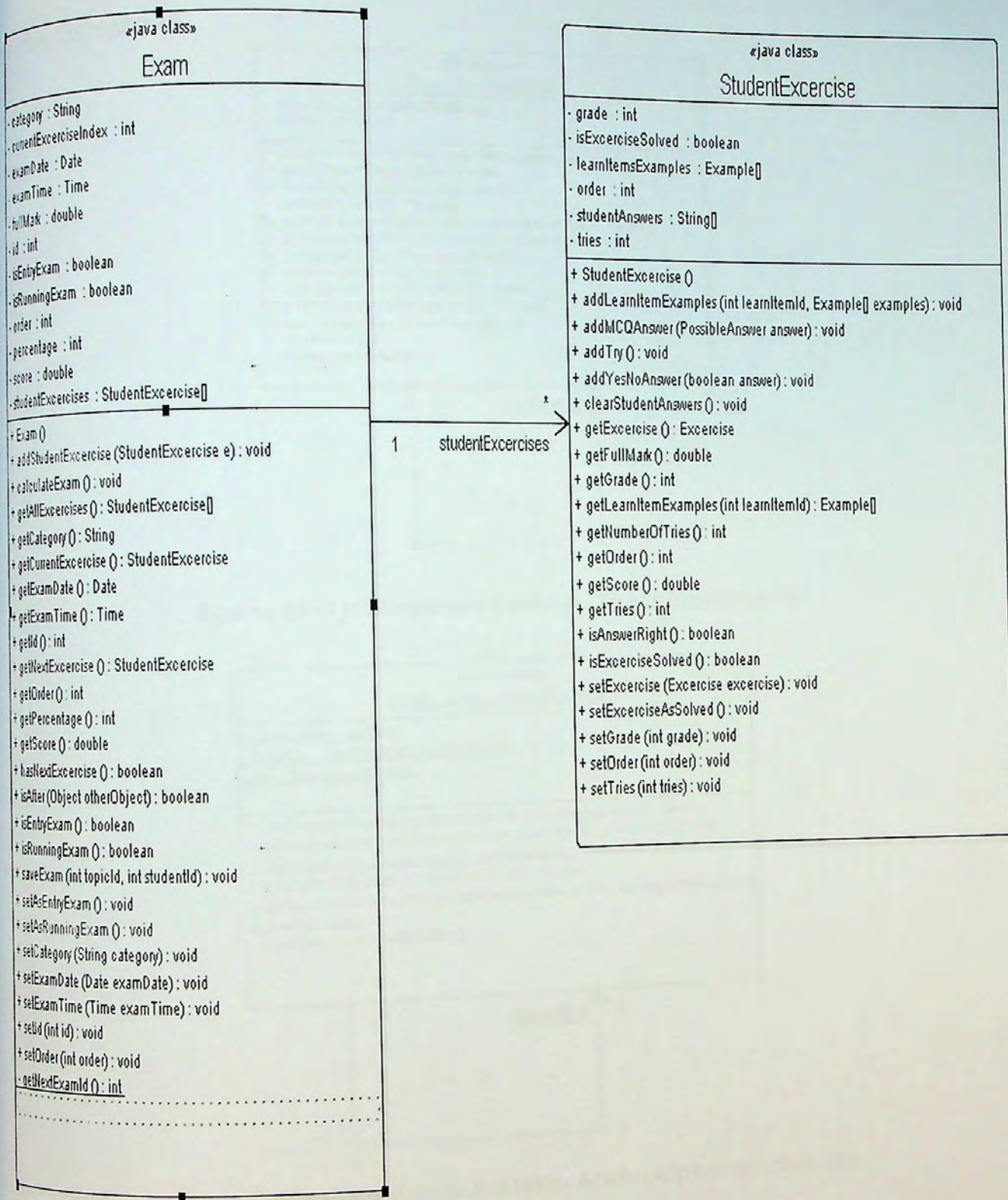


Figure (F-2): Whole Part Detailed Class Diagram Exam-StudentExercise

Pattern 2: Singleton

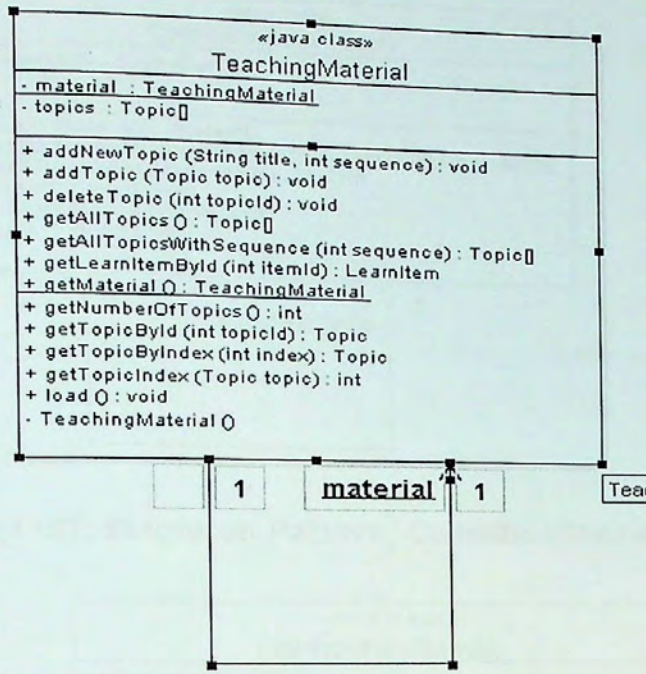


Figure (F-3): Singleton Pattern. TeachingMaterial

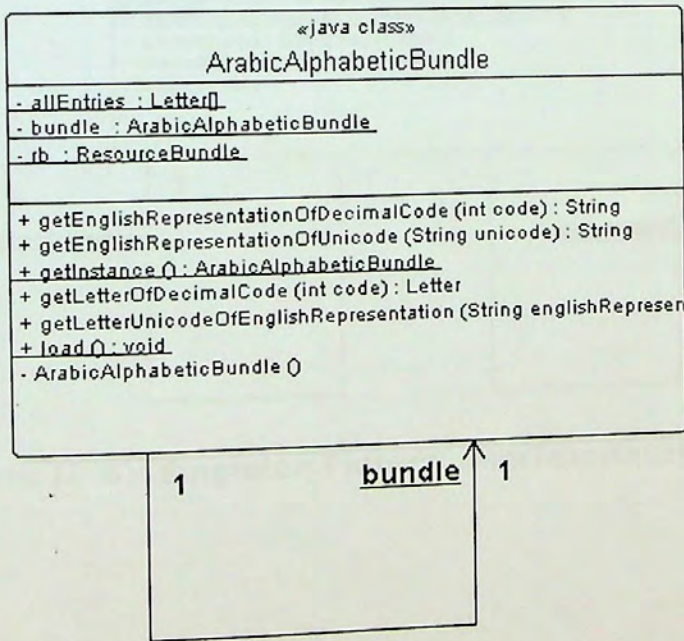


Figure (F-4): Singleton Pattern. ArabicAlphabeticBundle

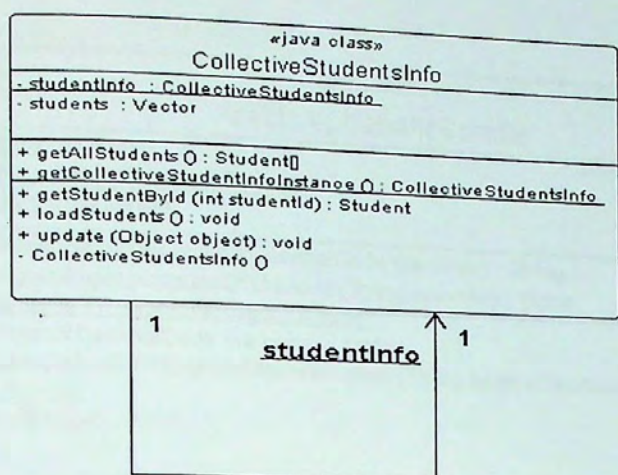


Figure (F-5): Singleton Pattern. CollectiveStudentInfo

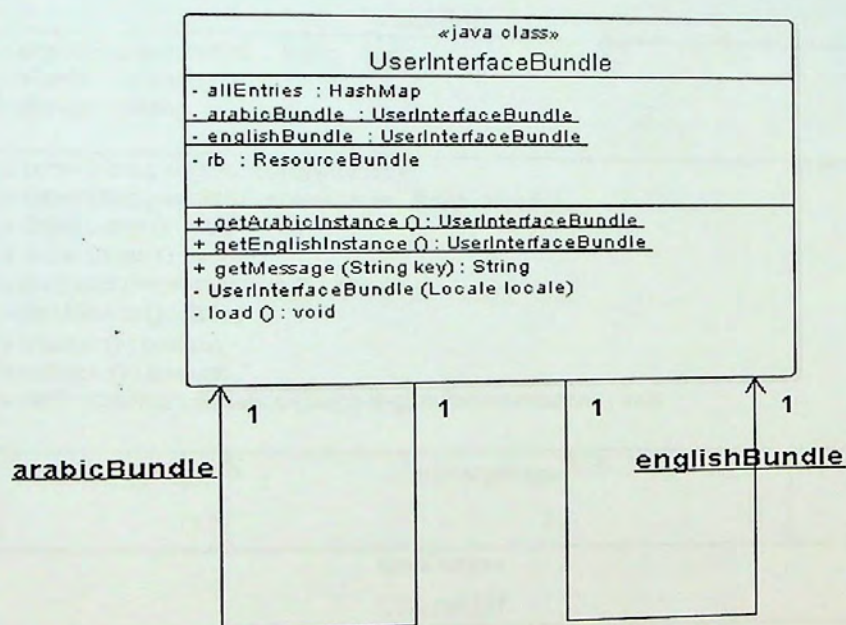


Figure (F-6): Singleton Pattern. UserInterfaceBundle

Figure 3: New Flyweight

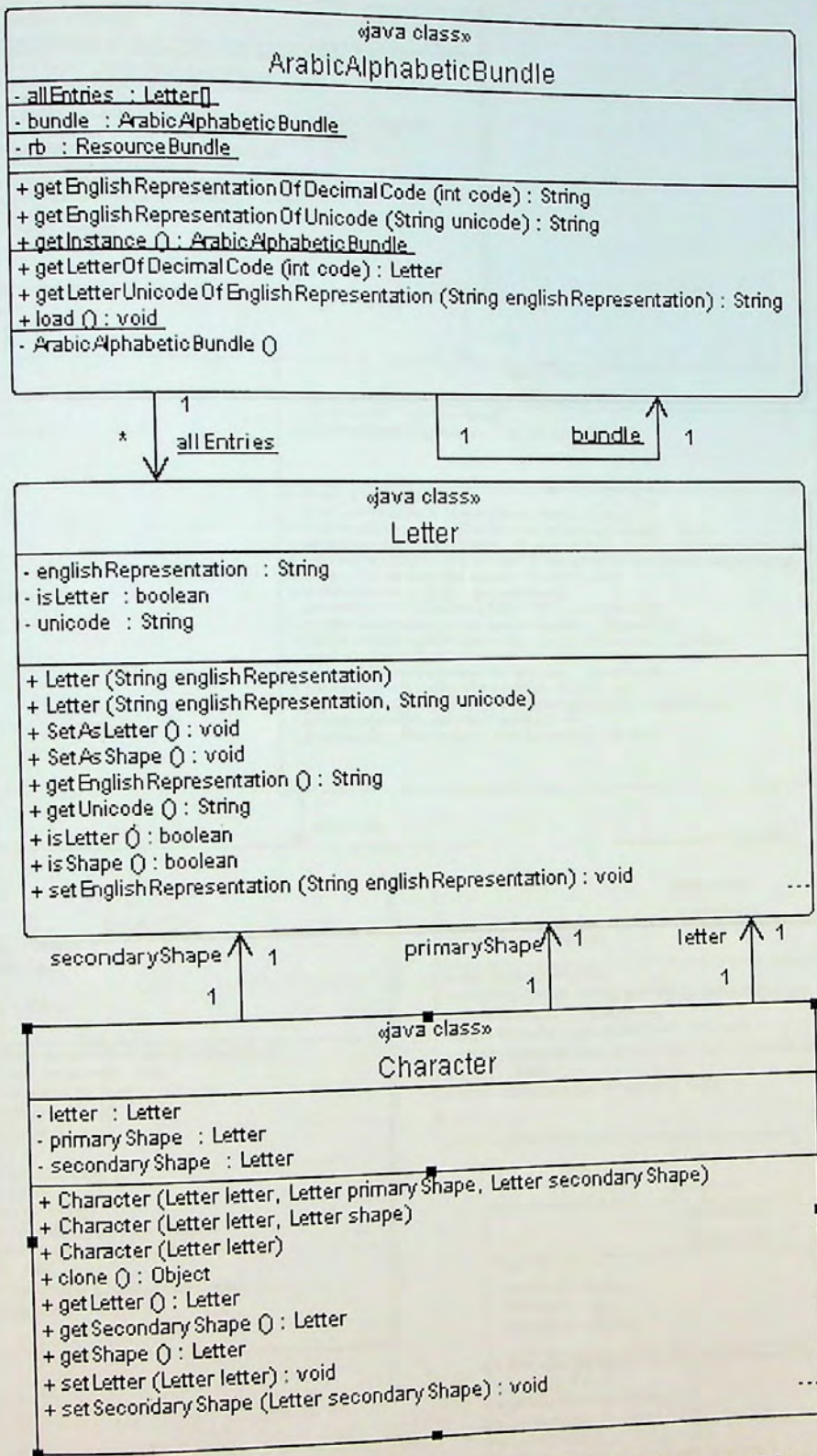


Figure (F-7): New Flyweight Pattern Detailed Class Diagram

ArabicAlphabeticBundle-Letter-Character



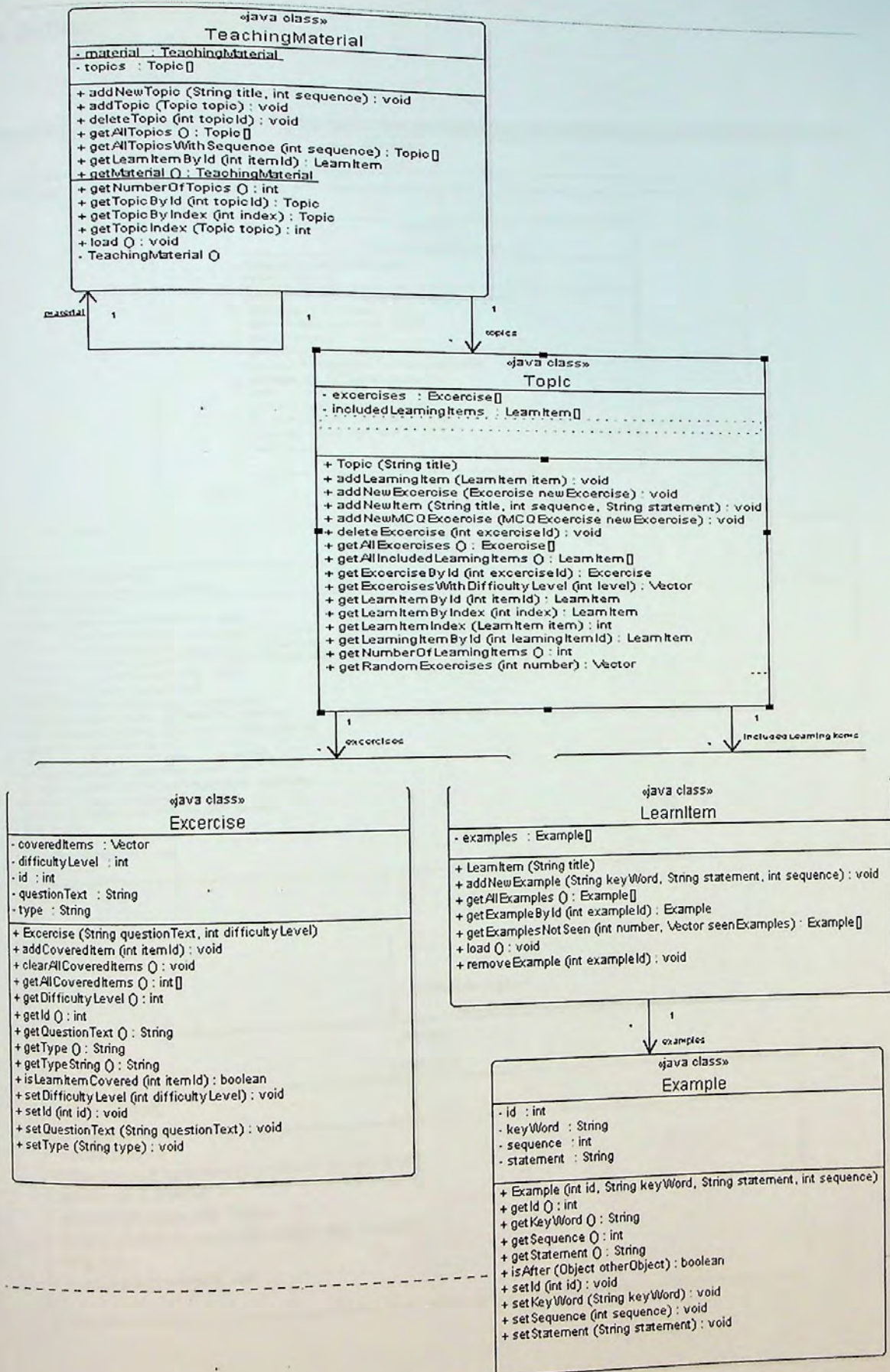


Figure (F-8): New Flyweight Pattern Detailed Class Diagram

TeachingMaterial-Topic-LearnItem-Exercise-Example

Pattern 4: Builder

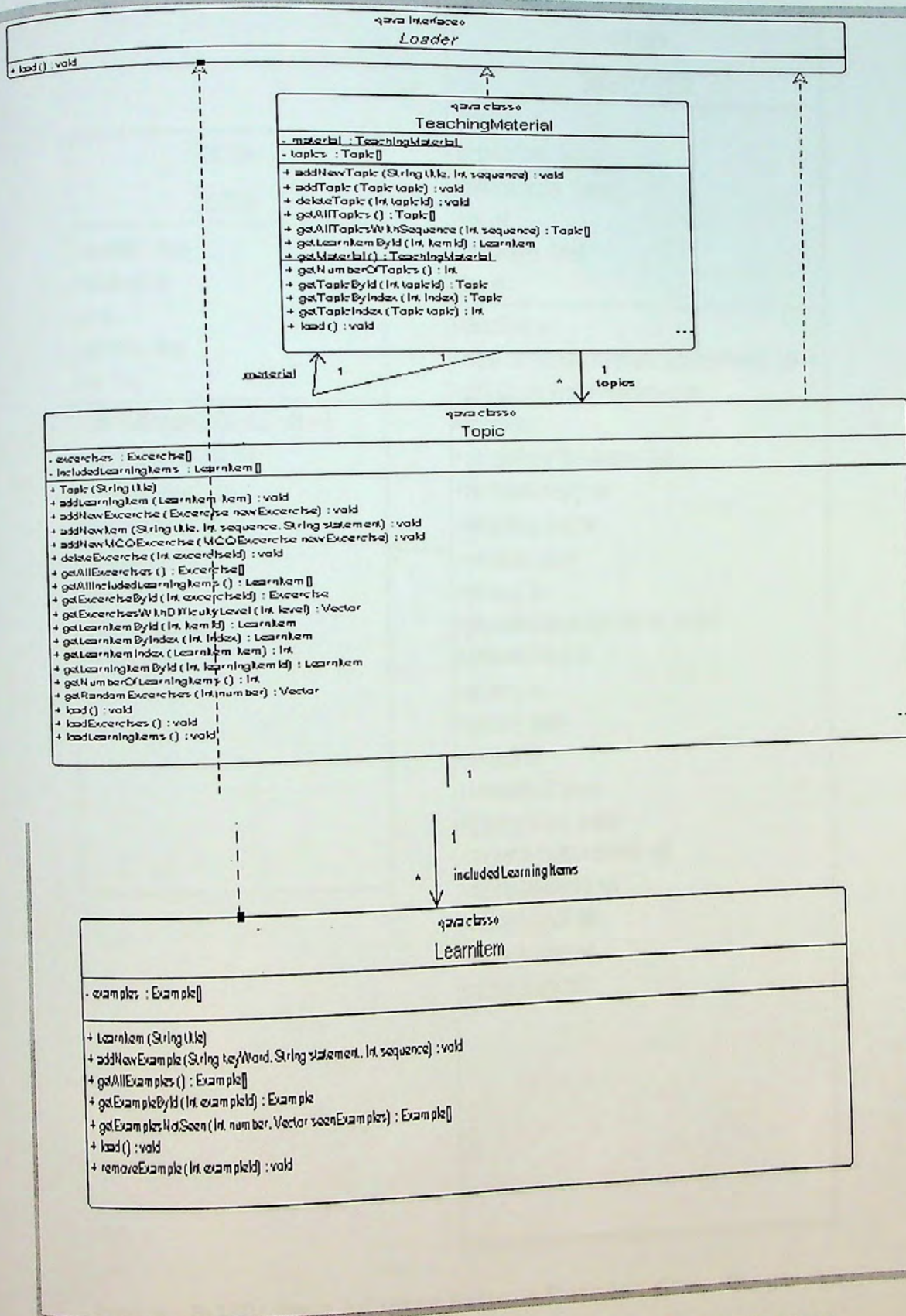


Figure (F-9): Builder Pattern Detailed Class Diagram

Pattern 5: New Adapter Pattern

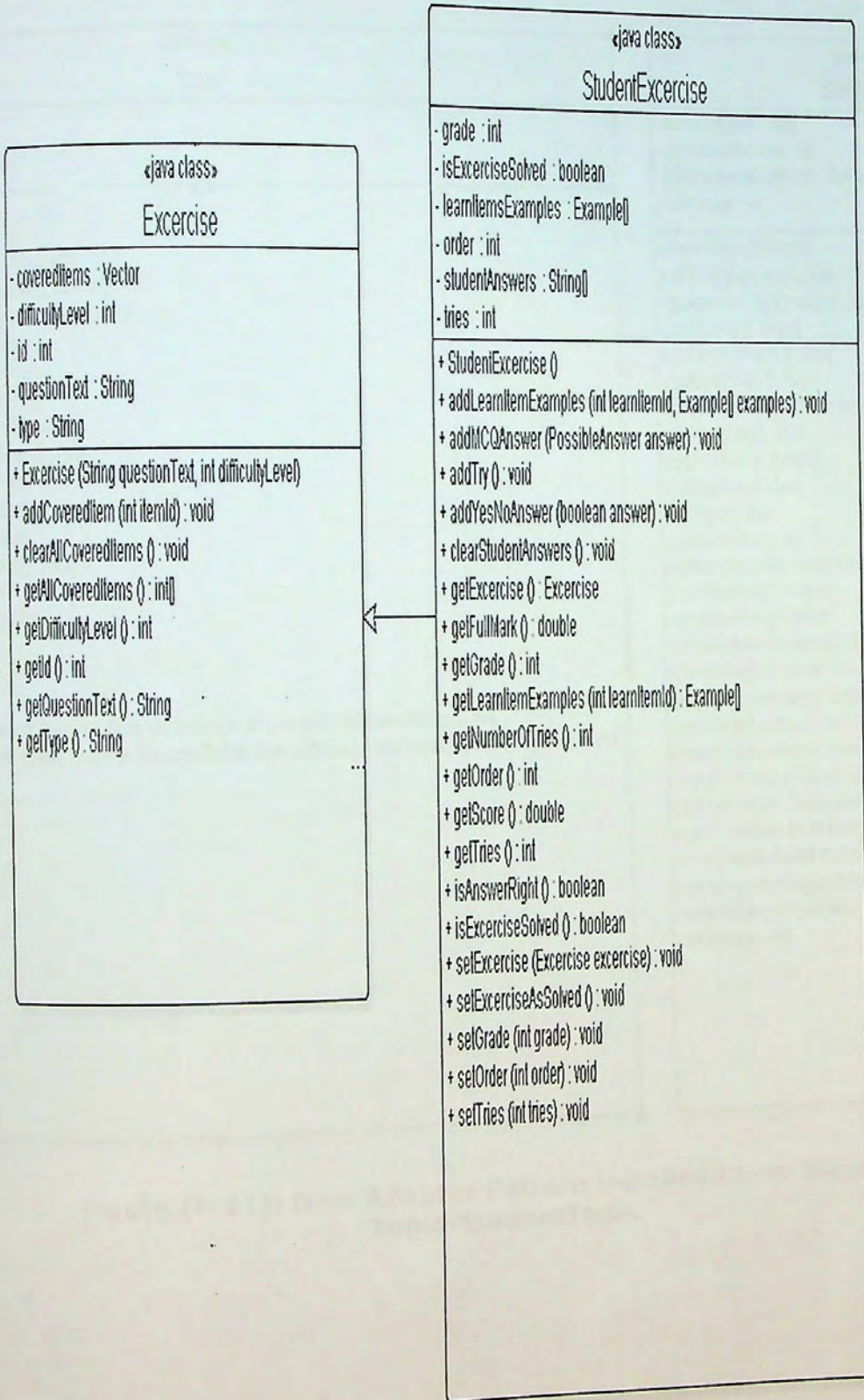


Figure (F-10): New Adapter Pattern Detailed Class Diagram

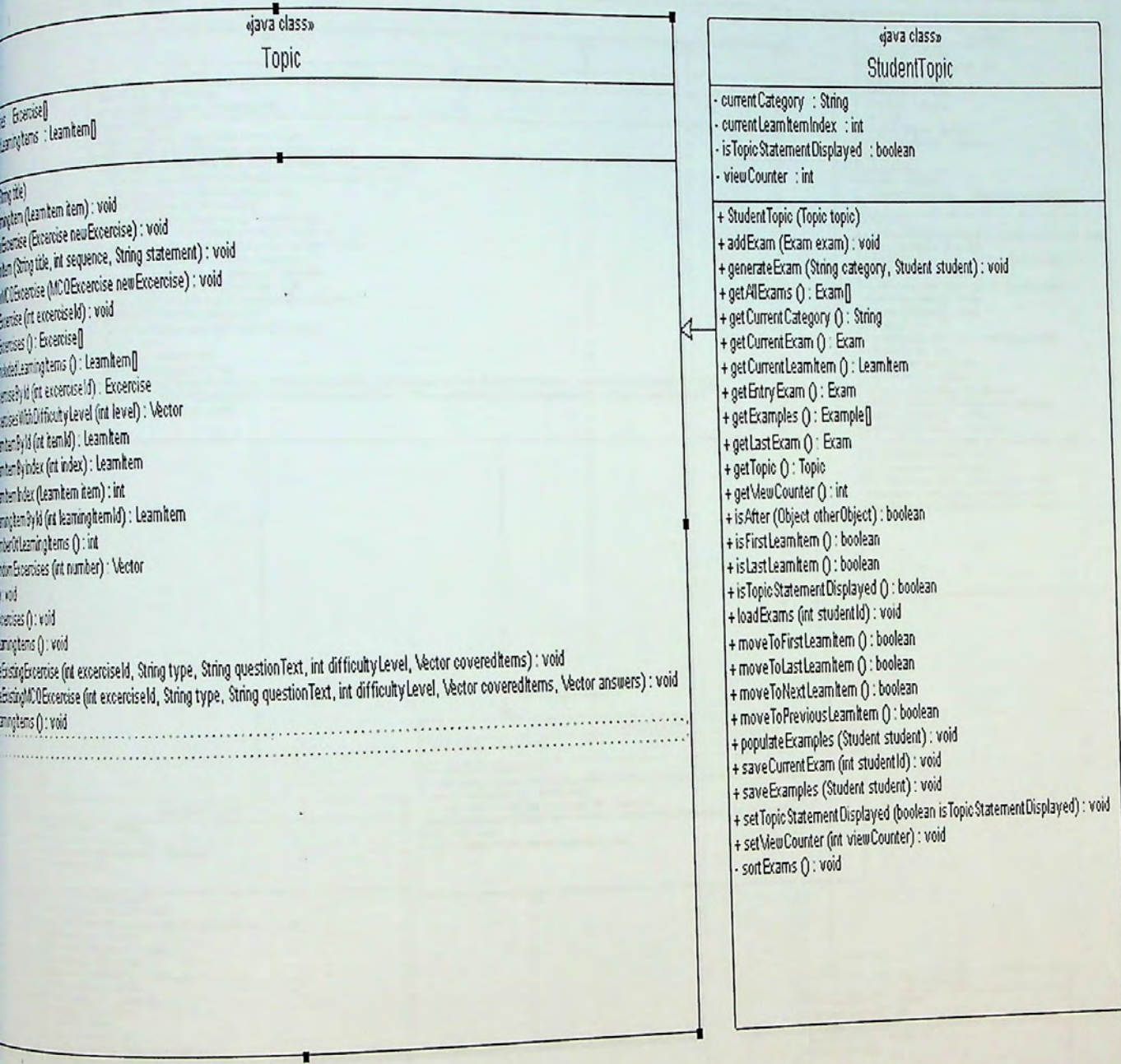


Figure (F-11): New Adapter Pattern Detailed Class Diagram Topic-StudentTopic

Pattern 6: New Strategy

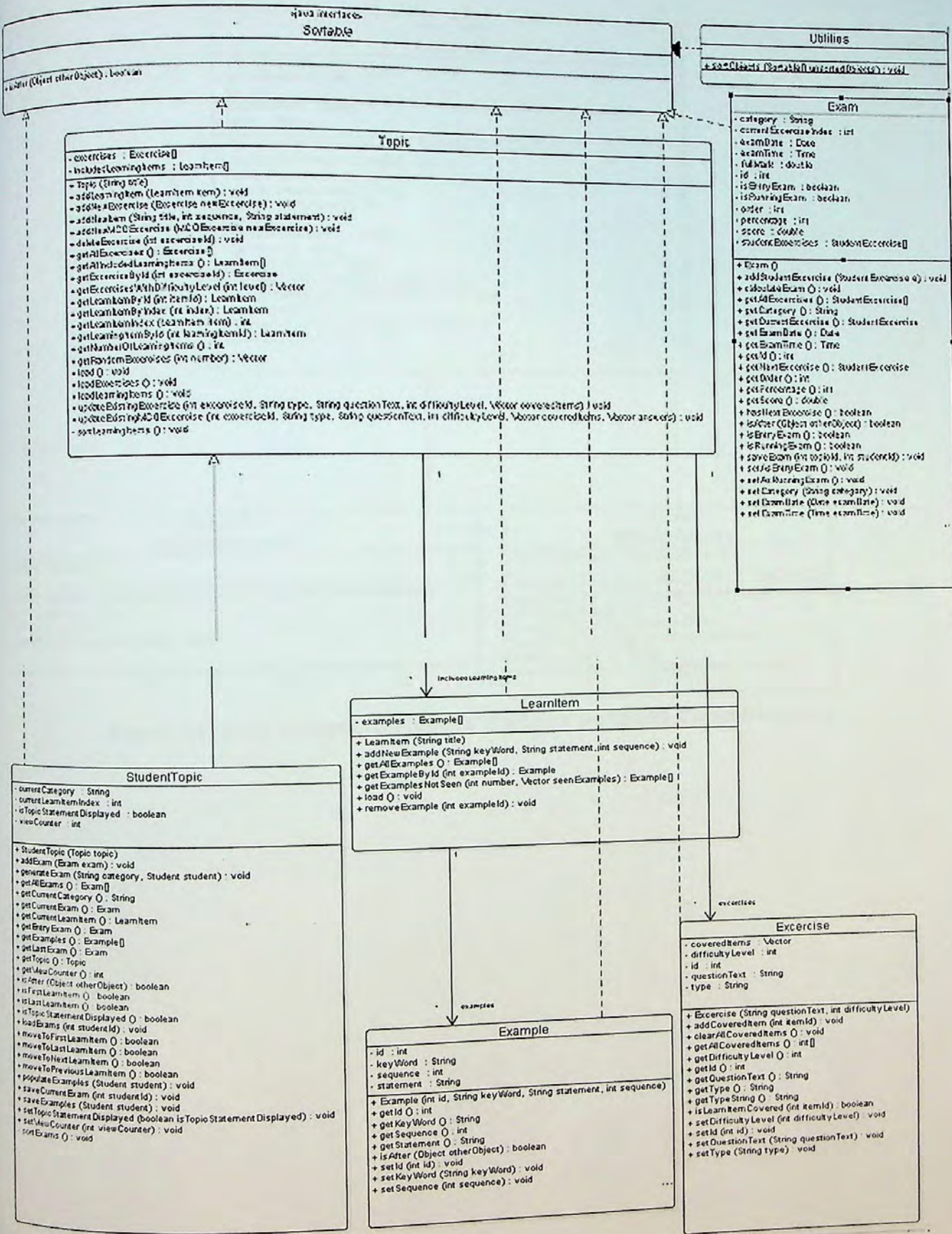


Figure (F-12): New Strategy Pattern Detailed Class Diagram

Pattern 7: Template Method

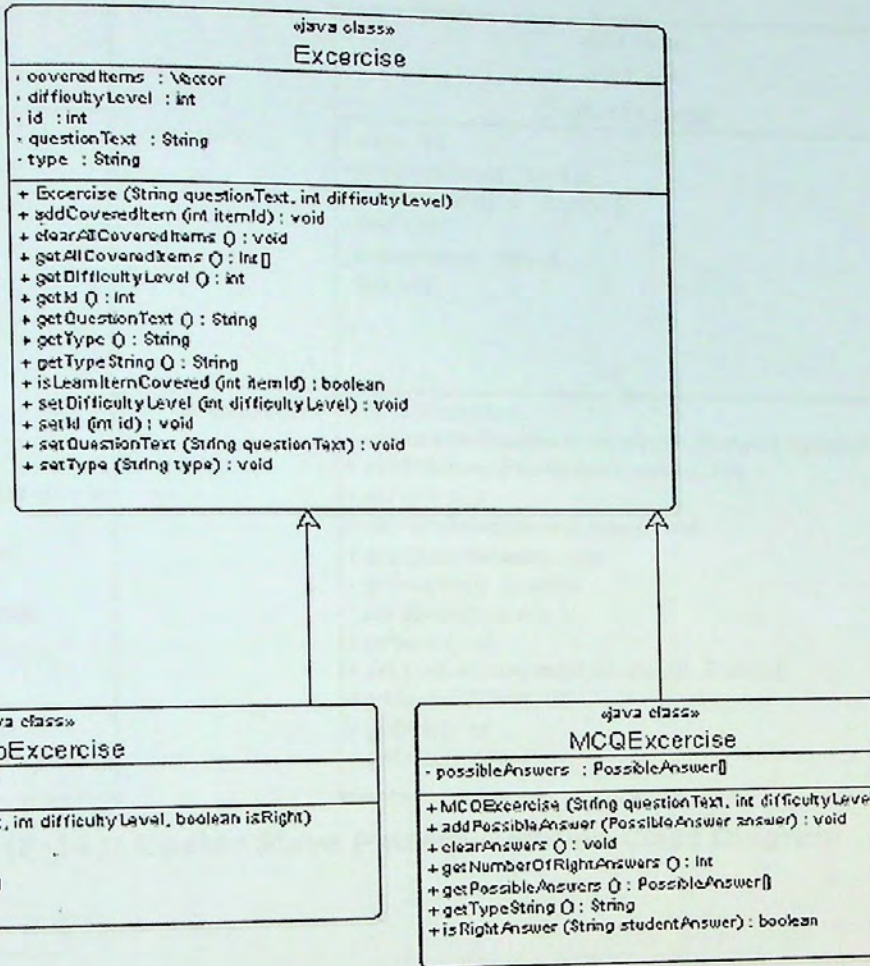


Figure (F-13): Template Method Pattern Detailed Class Diagram

Pattern 8: Master Slave

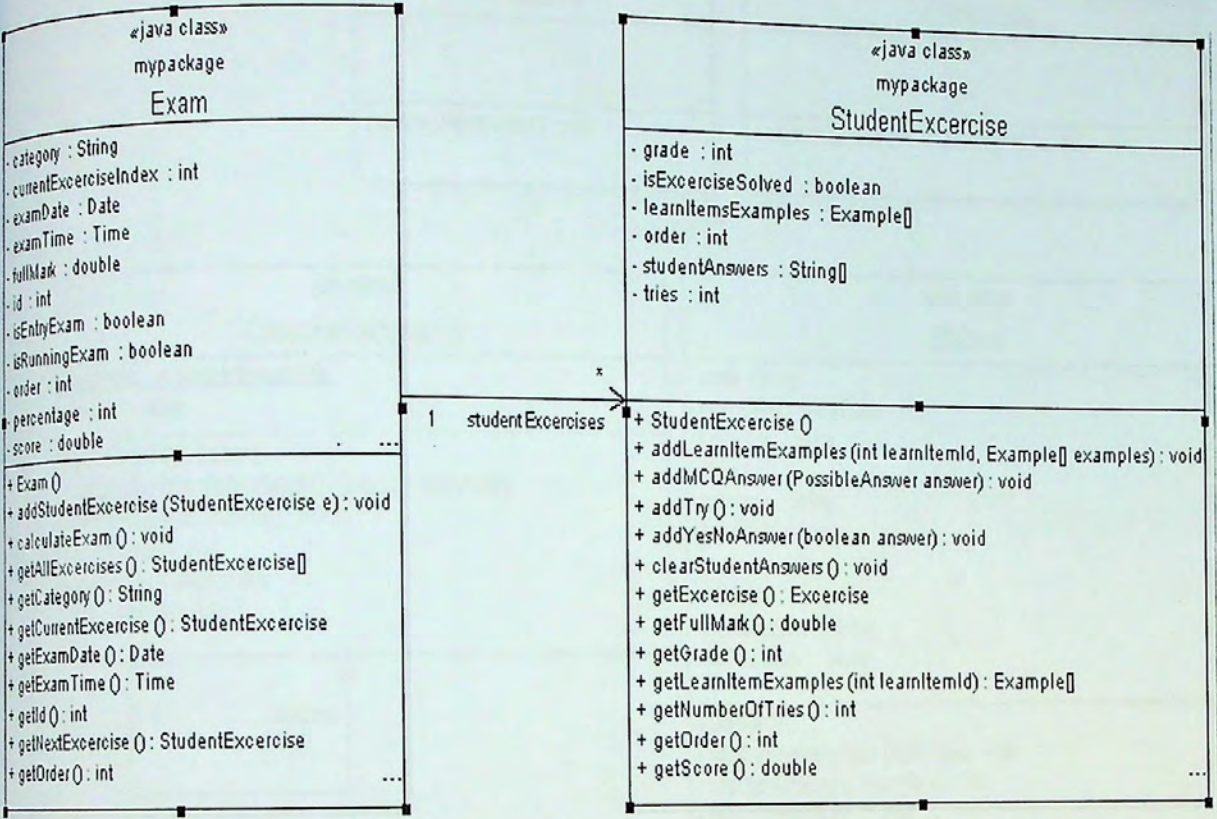


Figure (F-14): Master Slave Pattern Detailed Class Diagram

Pattern 9: Observer

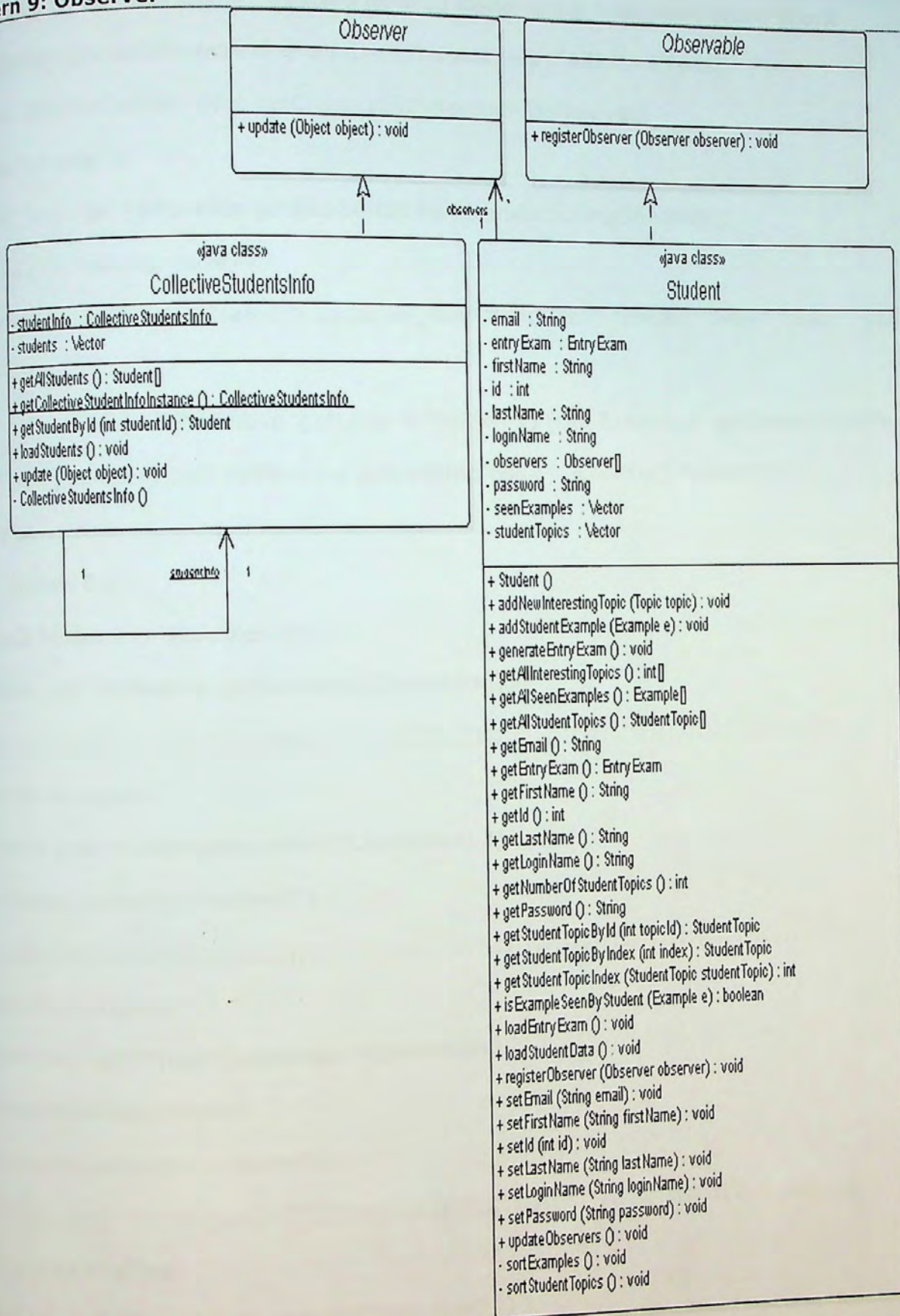


Figure (F-15): Observer Pattern Detailed Class Diagram



**Appendix G: Sample Expert System Source Code using Jess for Future Work**

```
(deffunction get-sa2altomoneeha-number-of-characters (?infinitive ?past)
  (bind ?difference (call ?past getCharactersDifference ?infinitive))
  (bind ?counter 1)
  (bind ?size (call ?difference getNumberOfCharactersExcludingShaddah))
  (while (<= ?counter ?size)
    (bind ?words (create$ "seen" "hamza_on_alef" "lam" "teh" "meem" "waw" "noon" "yeh" "heh"
"alef"
      (call ?infinitive getLetterAtPosition 2) (call ?infinitive getLetterAtPosition 3) ) )
    (if (not (member$ (call ?difference getLetterAtPosition ?counter) ?words) )
      then
        (return 0))
      (bind ?counter (+ ?counter 1)))
  (return (call ?difference getNumberOfCharacters)))
```

.....

```
(defrule is-mazeed
  (test (> (call ?*past* getNumberOfCharacters) 3) )
  =>(assert (verb-type mazeed)))
```

.....

```
(defrule is-mogarad
  (test (eq (call ?*past* getNumberOfCharacters) 3) )
  (verb-type ?x&~mazeed)
  =>(assert (verb-type mogarad)))
```

.....

```
(defrule wazn-af3ala
  (test (eq (call ?*past* getNumberOfCharacters) 4) )
  (test (eq (call ?*past* findLetter "hamza_on_alef") 1))
  (test (eq (call ?*past* getLetterAtPosition 2) (call ?*infinitive* getLetterAtPosition 1)))
```

```
(test (eq (call ?*past* getShapeAtPosition 2) "sukun"))
(test (eq (call ?*past* getLetterAtPosition 3) (call ?*infinite* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 4) (call ?*infinite* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 4) "fatha" )
=>(assert (past-wazn af3ala)))
```

.....

```
(defrule wazn-fa3ala
(test (eq (call ?*past* getNumberOfCharacters) 4) )
(test (eq (call ?*past* getLetterAtPosition 1) (call ?*infinite* getLetterAtPosition 1)))
(test (eq (call ?*past* getShapeAtPosition 1) "fatha"))
(test (eq (call ?*past* findLetter "alef") 2))
(test (eq (call ?*past* getLetterAtPosition 3) (call ?*infinite* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 4) (call ?*infinite* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 4) "fatha"))
=>(assert (past-wazn fa3ala)))
```

.....

```
(defrule wazn-fa33ala
(test (eq (call ?*past* getNumberOfCharacters) 4) )
(test (eq (call ?*past* getLetterAtPosition 1) (call ?*infinite* getLetterAtPosition 1)))
(test (eq (call ?*past* getShapeAtPosition 1) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 2) (call ?*infinite* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 2) "fatha"))
(test (eq (call ?*past* getSecondaryShapeAtPosition 2) "shadda"))
(test (eq (call ?*past* getLetterAtPosition 3) (call ?*infinite* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha" )
=>(assert (past-wazn fa33ala)))
```

```

.....
(defrule wazn-efa3ala
(test (eq (call ?*past* getNumberOfCharacters) 5) )
(test (eq (call ?*past* findLetter "alef") 1))
(test (eq (call ?*past* findLetter "noon") 2))
(test (eq (call ?*past* getShapeAtPosition 2) "sukun"))
(test (eq (call ?*past* getLetterAtPosition 3) (call ?*infinitive* getLetterAtPosition 1)))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 4) (call ?*infinitive* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 4) "fatha" )
(test (eq (call ?*past* getLetterAtPosition 5) (call ?*infinitive* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 5) "fatha" )
=>(assert (past-wazn enfa3ala)))

```

```

.....
(defrule wazn-ef3ala
(test (eq (call ?*past* getNumberOfCharacters) 5) )
(test (eq (call ?*past* findLetter "alef") 1))
(test (eq (call ?*past* findLetter "teh") 3))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 2) (call ?*infinitive* getLetterAtPosition 1)))
(test (eq (call ?*past* getShapeAtPosition 2) "sukun"))
(test (eq (call ?*past* getLetterAtPosition 4) (call ?*infinitive* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 4) "fatha" )
(test (eq (call ?*past* getLetterAtPosition 5) (call ?*infinitive* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 5) "fatha" )
=>(assert (past-wazn efa3ala)))

```

```

.....
(defrule wazn-ef3alla

```



```
(test (eq (call ?*past* findLetter "alef") 3))
(test (eq (call ?*past* getLetterAtPosition 2) (call ?*infinitive* getLetterAtPosition 1)))
(test (eq (call ?*past* getShapeAtPosition 2) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 4) (call ?*infinitive* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 4) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 5) (call ?*infinitive* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 5) "fatha" )
=>(assert (past-wazn tafa3aal)))
```

.....

```
(defrule wazn-estaf3ala
(test (eq (call ?*past* getNumberOfCharacters) 6) )
(test (eq (call ?*past* findLetter "alef") 1))
(test (eq (call ?*past* findLetter "seen") 2))
(test (eq (call ?*past* getShapeAtPosition 2) "sukun"))
(test (eq (call ?*past* findLetter "teh") 3))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 4) (call ?*infinitive* getLetterAtPosition 1)))
(test (eq (call ?*past* getShapeAtPosition 4) "sukun"))
(test (eq (call ?*past* getLetterAtPosition 5) (call ?*infinitive* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 5) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 6) (call ?*infinitive* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 6) "fatha" )
=>(assert (past-wazn estaf3ala)))
```

.....

```
(defrule wazn-ef3aw3ala
(test (eq (call ?*past* getNumberOfCharacters) 6) )
(test (eq (call ?*past* findLetter "alef") 1))
(test (eq (call ?*past* getLetterAtPosition 2) (call ?*infinitive* getLetterAtPosition 1)))
```

```
(test (eq (call ?*past* getShapeAtPosition 2) "sukun"))
(test (eq (call ?*past* getLetterAtPosition 3) (call ?*infinitive* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* findLetter "waw") 4))
(test (eq (call ?*past* getShapeAtPosition 4) "sukun"))
(test (eq (call ?*past* getLetterAtPosition 5) (call ?*infinitive* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 5) "fatha"))
(test (eq (call ?*past* getLetterAtPosition 6) (call ?*infinitive* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 6) "fatha"))
=>(assert (past-wazn ef3aw3ala)))
```

.....

```
(defrule wazn-ef3aaalla
(test (eq (call ?*past* getNumberOfCharacters) 6) )
(test (eq (call ?*past* findLetter "alef") 1))
(test (eq (call ?*past* getLetterAtPosition 2) (call ?*infinitive* getLetterAtPosition 1)))
(test (eq (call ?*past* getShapeAtPosition 2) "sukun"))
(test (eq (call ?*past* getLetterAtPosition 3) (call ?*infinitive* getLetterAtPosition 2)))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* findLetter "alef") 4))
(test (eq (call ?*past* getLetterAtPosition 5) (call ?*infinitive* getLetterAtPosition 3)))
(test (eq (call ?*past* getShapeAtPosition 3) "fatha"))
(test (eq (call ?*past* getSecondaryShapeAtPosition 3) "shadda"))
=>(assert (past-wazn ef3aaalla)))
```

.....

```
(defrule mazeed-1
(or
(past-wazn af3ala)
(past-wazn fa3ala)
```

AMERICAN UNIV. IN CAIRO LIBRARY  
3 8534 01214 2083