Middle East Technical University
Department of Computer Engineering

Initial Design Report

PAPAĞAN

by

Korsan Yazılım

December, 2005

Contents

1 Introduction

1.1 Purpose

The purpose of this document is to describe *how* to map the complete requirements of Papağan Education Tool into software architecture.

1.2 Scope

The initial design of the project focuses on critical implementation priorities and establishes a base for further design processes. A critical implementation priority guides to a task that has to be done right for the success of the overall project [1]. Therefore, beginning from the highest abstraction level of the system, this phase decreases the abstraction by covering the engineering details of each level until there is nothing to be left unclear for implementing a practical solution for the problem. However, some design details are left to detailed design documentation and there may be additions or deletions to this document in the flow of the design process.

1.3 Description

In the analysis phase, we have stated the problem and specified higher level description of Papağan. We have tried to capture requirements from the customers' perspective indepented of how these requirements will be accomplished.

On the other hand, the design document is a blueprint for the implementation of the system. In this document, after giving introductory information about our initial design report, we continue with design considerations; the guidelines, aims and general limitations. Then, the very important part of our document comes, the system architecture. We try to describe the architecture from different interrelated views. The design phase should start decomposing from the highest level system capabilities and stop when key abstractions are simple enough to require no further decomposition. We document and visualize the decomposition process and each decomposed part of the system in the rest of the document.

2 Design Considerations

2.1 Assumptions and Dependencies

2.1.1 Reliability and Interoperability of Used Toolkits and Libraries

We use Ogre3D for animating model which is created using Blender3D. We assume that integration of Ogre3D components to our system is not problematic.

2.1.2 Portability

Papağan is platform dependent and executable only under Microsoft .NET framework.

2.1.3 Performance

3D animations in Papağan will need considerable amount of processing for rendering animations at the rate of 25 to 30 frames per second.

2.2 Development Methods

In general, we partition the problem into simpler subproblems, each of which can be considered independently (Divide and Conquer); delay some decisions considered as details to focus on more crucial topics (Stepwise Refinement); and begin with the most general aspects of the problem which use other components (Top Down).

More specifically, we have an object oriented approach to design phase. An object oriented design (OOD) is use case driven and architecture-centric method. It can provide us benefits of reusability and maintainability. OOD can be thought of in two phases. The first, high-level design, deals with the decomposition of the system into large, complex objects. The second phase, low-level design, attributes and methods are specified at the level of individual objects [2] .

We use UML for visual modeling. Because of being an use case driven and architecture-centric method, we can get most benefit from the UML by using it in OOD method.

2.3 Goals and Objectives

Our design goal is to achieve a well structured system that is functionally, logically and physically cohesive; formed of loosely coupled subsystems [3]. This type of system is helpful for us to visualize, specify, construct and document it from different, yet interrelated, aspects. Also, unavoidable changes can be tolerated easily when working on this kind of system.

3 System Architecture

Every system, even composed of one component, has architecture. Our main purpose for this phase is to create a suitable architecture for Papağan. Mainly, our architecture defines major components of Papağan with relations (structure) and interactions between these components.

3.1 Papağan Architecture Overview

Data Store handles file operations and database queries. This package interacts with all the packages that need to execute file operations or database queries.

Editor is able to create or edit animations. This package interacts with Data Store for the file or database operations. It also interacts with GUI for the user I/O.

Vocabulary Lessons is able to create or edit vocabulary based lessons. It contains a specialized viewer for these lessons. This package interacts with Data Store for the file or database operations. It also interacts with GUI for the user I/O.

Games include a vocabulary game called Initials. This package interacts with Data Store for the file or database operations. It also interacts with GUI for the user I/O.

Dictionary includes a categorized lexicon of TİD expressions. It is also supported with additional materials such as pictures and videos. This package interacts with Data Store for the file or database operations. It also interacts with GUI for the user I/O.

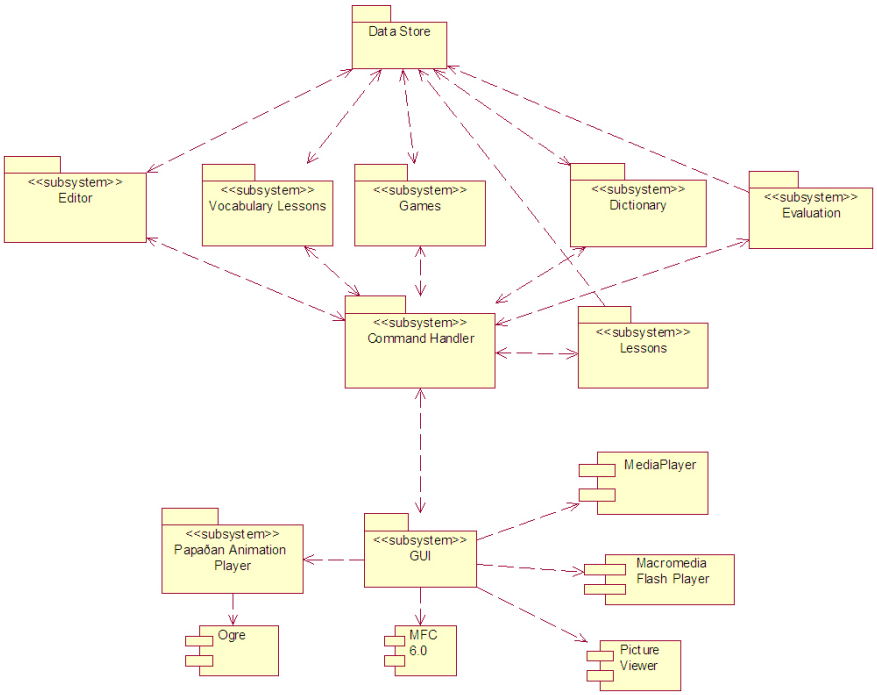

Figure 1: Structure

Evaluation package is able to create and evaluate multiple choice exams. This package interacts with Data Store for the file or database operations. It also interacts with GUI for the user I/O.

Lessons include flash animations for TİD education. This package interacts with Data Store for the file or database operations. It also interacts with GUI for the user I/O.

Papağan Animation Player includes a player for the 3D animations of TİD expressions. It interacts with Ogre3D external entity to visualize the animations.

GUI interacts with all of the packages except Data Store. It is the layer between the user and other packages that needs to communicate with the user. This package also interacts with the external entities MFC 6.0, Media Player, Macromedia Flash Player, and Picture Viewer.

4 Detailed System Design

4.1 Graphical User Interface

- *Dictionary*

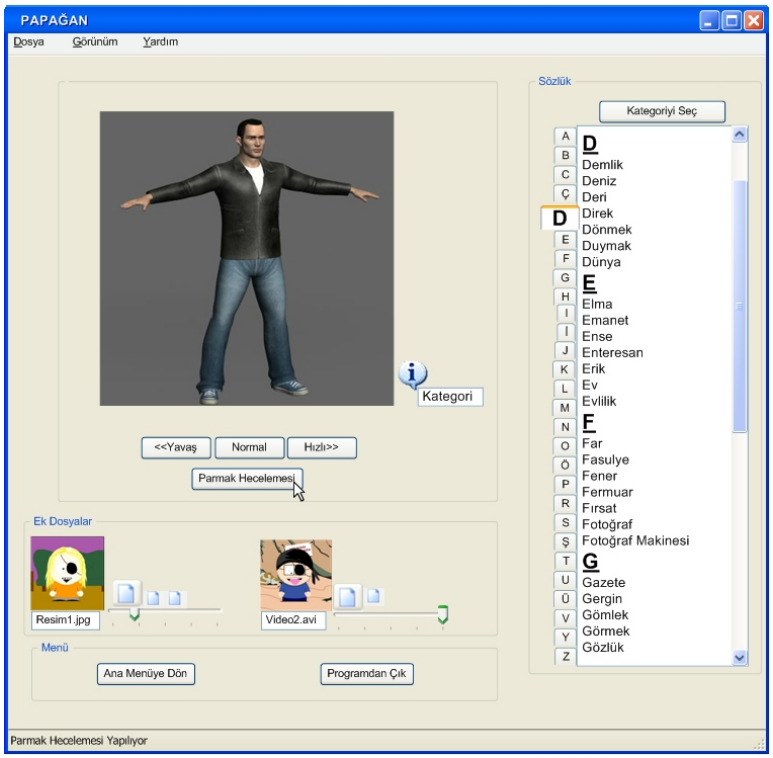

Figure 2: Dictionary GUI

Dictionary GUI [Figure 2] is mainly based on two main parts. On the right of the screen, user can select the word that he wonders to see its matching animation. After selection, Papağan Animation Player displays the selected word's animation on the left of the screen. User has the chance to arrange the speed of the animation with the help of the buttons, placed right below the player. Moreover user can choose to view the fingerspell of the word. The categories that contains the

selected course is viewed on the right hand side of the player, also the pictures and the videos related with the word can be displayed on the menu, placed at the bottom. User can access main menu and quit the program directly from the menu. Furthermore, all the abilities that a usual window has are added.

- *VocabularyLessonEditor*

Figure 3: Vocabulary Lesson Editor GUI

This interface [Figure 3] is designed to reduce the effort to build a vocabulary training session. User selects the word to be added from the right end of the screen. The player placed on the left side of the screen, displays the wanted animation. The tiny menu on the top right corner helps us to assign the title, to either the interface or accordingly the lesson. Below the player, the usual video displaying buttons can be observed. They can rewind, fast forward, play and pause the animation. Moreover, the menu below the buttons let us to determine what videos and pictures will be included. The menu on the right bottom corner, displays the words that are already added to the lesson. Like any other, a common menu and usual window attributes are added.

- *VocabularyLesson*

  Documents that are made with the VocabularyLessonEditor part are displayed with this tool [Figure 4]. End user selects the title from top right corner and starts to display the selected course. User can also observe the words throughout the course on the menu below. The animation, the photos and the videos can be displayed from the menus given. Like any other interfaces the common parts are included.

- *Animation Editor*

  This interface [Figure 5] is the tool for creating an animation manually, then attaching the required category, picture and video information and finally pushing the package into the dictionary. The menu on the top right corner helps us to choose matching categories, if it is not listed, it can also be added with menu's button. The menu below is for browsing the pictures and the videos of the word. The package is inserted in the database with the click on the right bottom button. The timeline on the bottom is used for expressing the animation. As all does, usual parts are added.

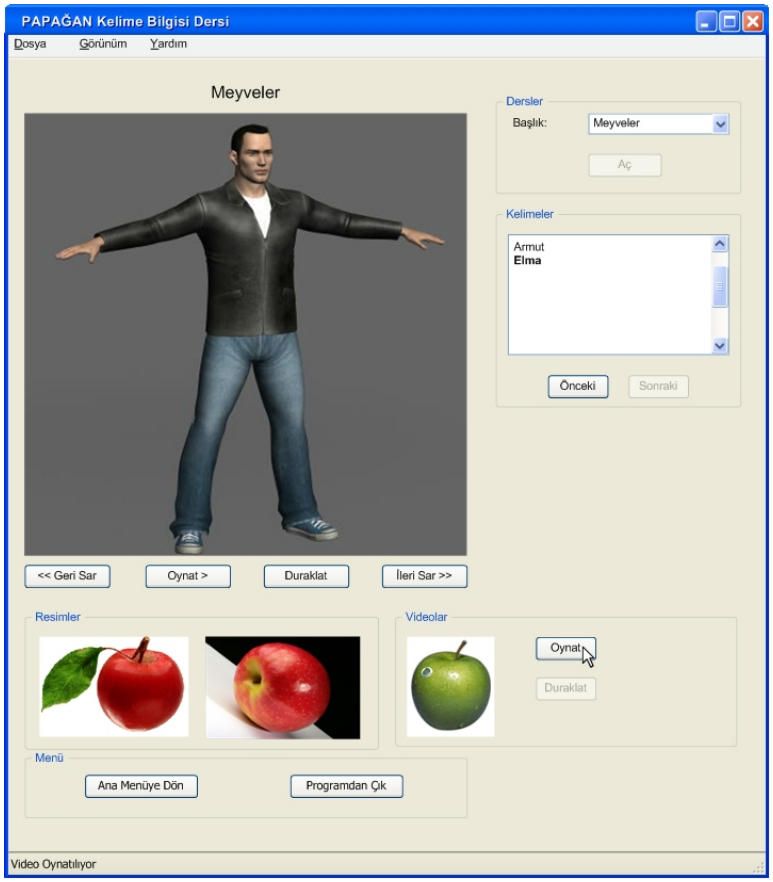

Figure 4: Vocabulary Lesson GUI

- *Exam Editor*

  Vocabulary exams are created via this interface[Figure 6]. On the right side of the screen, dictionary is listed for selecting the required words. Below this menu, we can adjust the time constraint and the scope of the exam. Additionally we can see what we added up to then and like all we meet the usual needs.

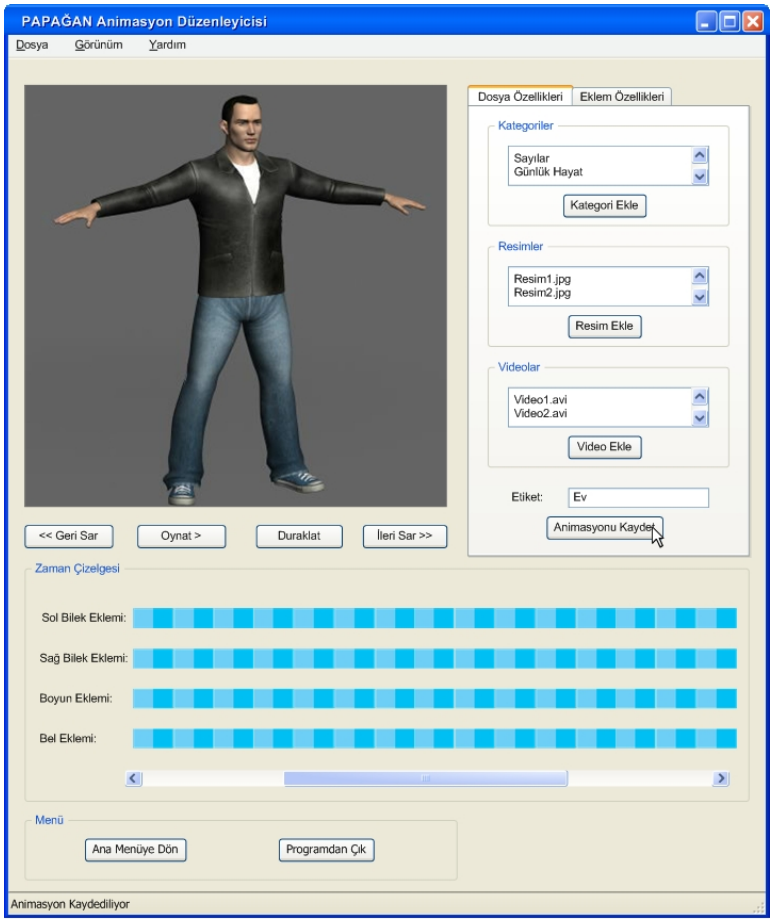

Figure 5: Animation Editor GUI

- *Exam*

  This interface[Figure 7] is for displaying the exam created by the Exam Editor. The animation and the buttons for it are listed on the top left corner. The answer can be given from the bottom menu or the answer shall not be given. Moreover the exam's name and the remaining time are also displayed on the top significantly.

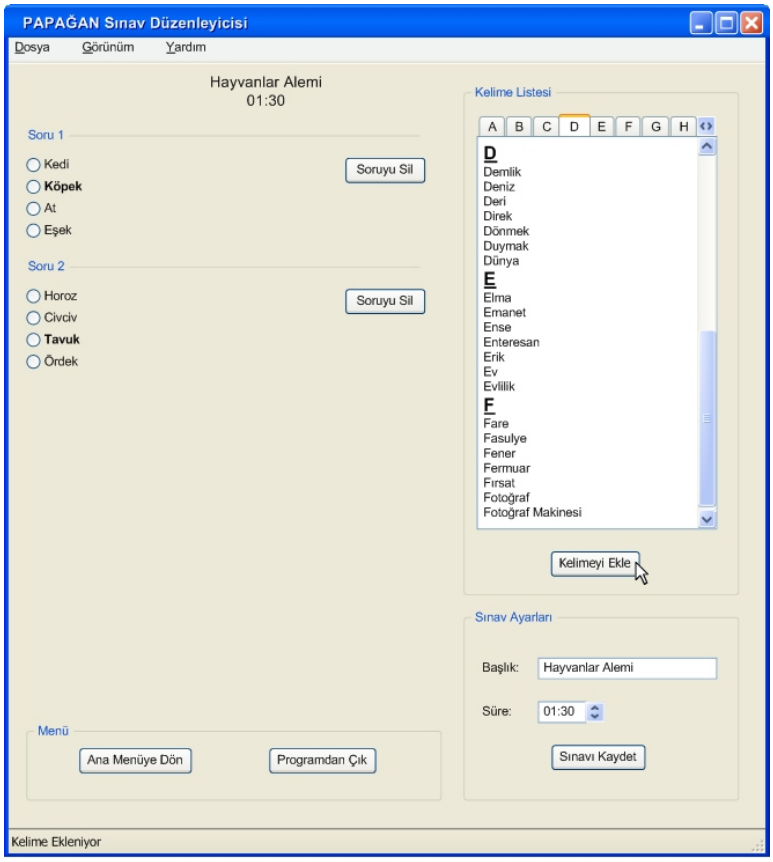

Figure 6: Exam Editor GUI

- *Lesson*

  This interface is just like any other animation player tool. We can display the exported animation, for example a Flash animation, with the buttons added to the menu.

- *Games*

  The game screen[Figure 9] involves the animation player on the left side. The letters are typed into the boxes supplied and on the bottom right corner, hint menu can be used. User shall change what he typed

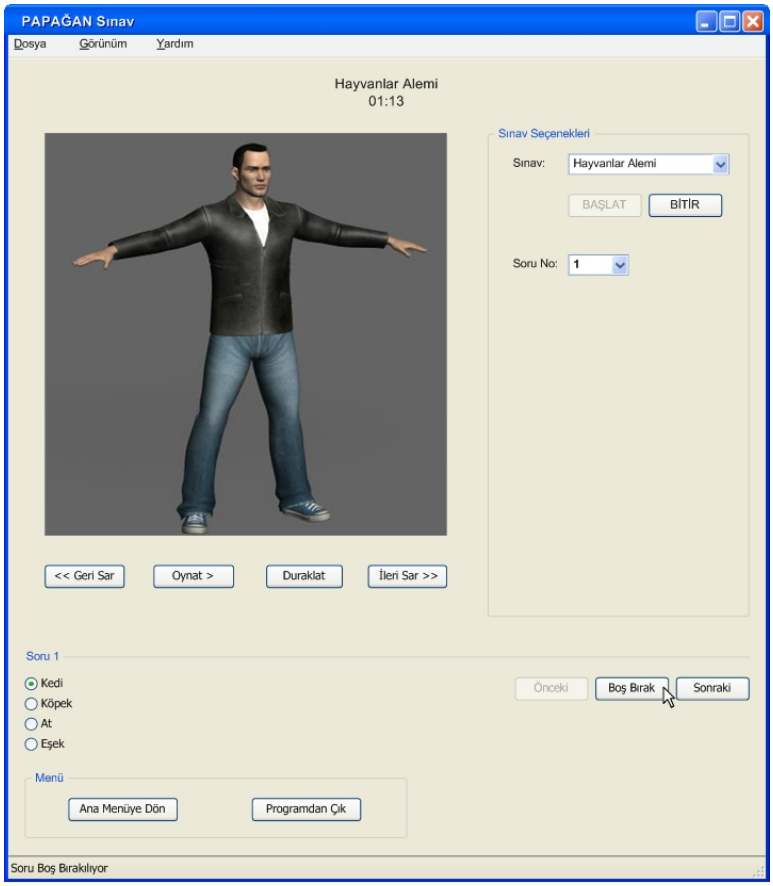

Figure 7: Exam GUI

until he presses the submit button. His current score also is displayed. Likely, the usual actions are listed in this interface.

4.2 Class Diagrams

4.2.1 Interface Classes

Interface classes are used in order to conduct user commands to the other related classes and retrieve the data and animations and visualize them. The methods implemented in these classes are triggered by user events such as selecting an item from a combo box, clicking a button or checking a box. Interface classes generally contain minimum data item for data only for visu-

Figure 8: Lesson GUI

alization and they use the other classes' methods in order to perform desired operations. The interaction with the interface objects (such as combo boxes , textboxes) are not described.

All interface classes are derived from the base class BaseInterface.

Base Interface

This is an abstract base class [Figure 10] which is used to derive the other interface classes in order to perform interface functionalities. BaseInter-

Figure 9: Games GUI

face contains minimum functionality which are common in all other derived classes . This functionalities can be stated as follows:

playAnimation() plays a sign animation activating the AnimationPlayer.

playFingerSpell() retrieves the sign animation of the word and plays it.

playVideo() plays video of the word calling related ActiveX object.

stopAnimation() stops the current animation play thread.

rewindAnimation() rewinds the animation back.

viewPicture() views the picture of a word.

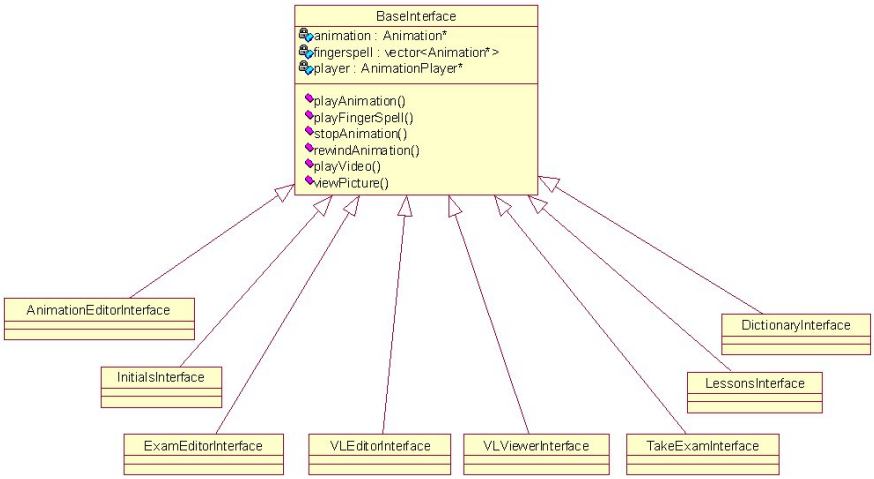

Figure 10: Base Interface

After describing the base class BaseInterface the description of interface classes can be expanded explaining the derived classes.

VLViewerInterface Class

VL stands for Vocabulary Lesson [Figure 11] for short. It has two protected attributes , one is the current lesson word (namely lessonWord) to display to the user The other is a pointer to a VLViewer object, which is used to fetch other words and related data.

As described above, the methods of VLViewerInterface mostly contain calls to vocabularyLesson object methods. They usually do not have parameters, but they retrieve parameters from user interface objects when calling vocabularyLesson methods.

setPanel() is sets the user interface components.

getVocabularyLessonList() is a method that invokes vocabulary lesson to fetch the names of available lessons list and displays them.

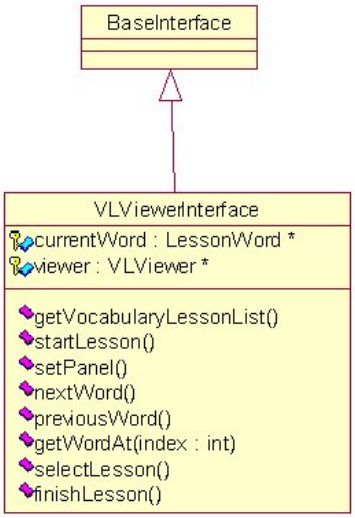

Figure 11: VLViewer Interface

A selected lesson can be started using startLesson(). nextWord(), previousWord() and getWordAt() methods provide navigation through the lesson. Animations,videos and pictures can also be viewed using base class facilities.

VLEditorInterface Class

VLEditorInterface [Figure 12] provides the interface to access the vocabulary lesson editor. It has a pointer to a VLEditor object, a vector of strings that contain current available categories in the database, the current word names related to selected category (if selected any, all dictionary otherwise), pointer to the currently selected word, and name of the current lesson.

The panel is set using the setPanel() function. Saving or loading a lesson, or clearing the content is done by the related methods.

User can invoke the methods of the VLEditor Object in order to navigate through the dictionary, add or remove lesson words to lesson;

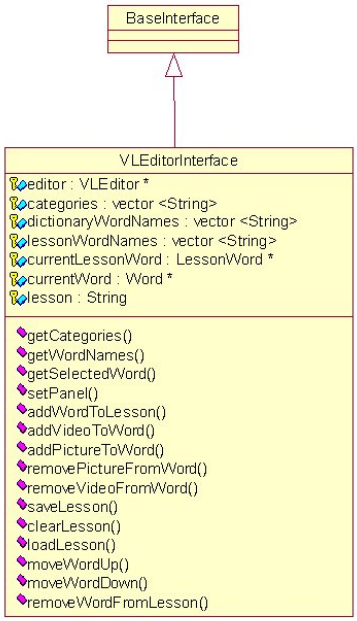

Figure 12: VLEditor Interface

add/remove pictures and videos to the lesson. The user can also move a word up and down in the lesson.

Also an animation or a video can be played or a picture can be viewed using base class facilities.

DictionaryInterface Class

Dictionary interface [Figure 13]is the way to interact with the dictionary of PAPAĞAN. It contains a pointer to a Dictionary object, a string vector containing the category names available in the database, and a pointer to the currently selected word as protected attributes.

The methods of the Dictionary object are invoked by the methods of this interface. getCategory() fetches the available categories whereas getWordNames() get the names of the words belonging to a category. getSe-

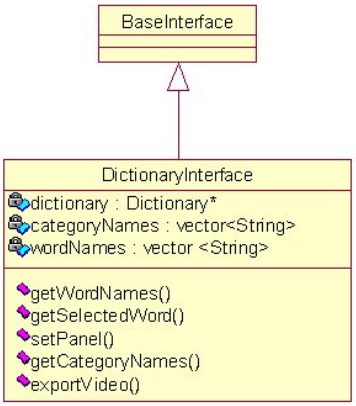

Figure 13: Dictionary Interface

lectedWord() retrieve the contents of the selected word. Also animation of a word can be exported to video using the exportVideo() function.

As usual, play animations,videos and pictures can be viewed using the base class facilities, setPanel() method adjusts the GUI components.

InitialsInterface Class

This class [Figure 14] provides interface to user to play initials. But it differs from the other interface classes in some way. The other interface classes completely separate the business logic from the interface. In initials interface, some operations are done in user the interface level since the interaction is with a single word only and, the timer used to keep time is in the user interface side.

This interface contains a timer that sets that used to terminate calculate score/terminate game. Also there is a pointer to the word which is currently asked to the user, and integer attributes keeping remaining time and current score. The categories of the question word are placed in a string vector

startInitials() starts the game and gets the first word, checkAnswer() evaluates the users answer and calls successfulAnswer(). If the answer is

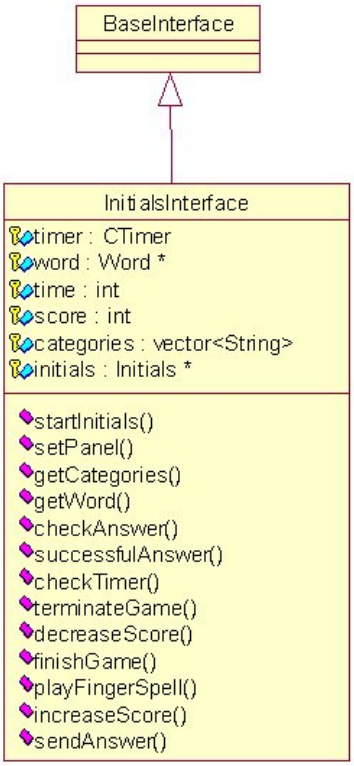

Figure 14: Initials Interface

correct score is updated and new word is fetched by getNewWord(). CTimer object timer invokes checkTimer() to check whether time is up if the time is up terminateGame() is activated. When a hint is displayed (namely a category of the word), decreaseScore() is invoked in order to adjust the score.

When the game is finished, finishGame() is called, if the score is higher than the score then the name of the player is recorded as the high scorer.

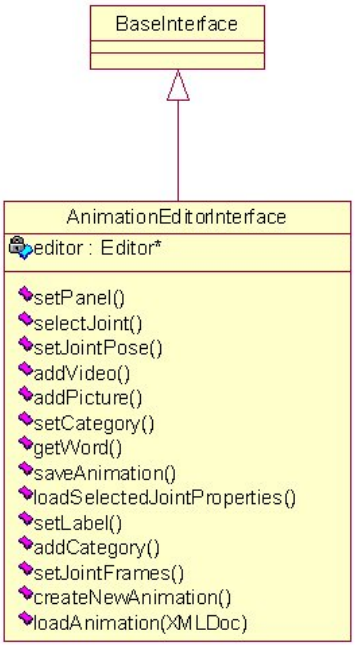

Figure 15: AnimationEditor Interface

AnimationEditorInterface Class

This interface [Figure 15] provides a way to edit animations. It has a pointer to the currently selected word, and currently selected animation. With selectJoint() user selects a join in cooperation with OGRE's event handler. User can add start/end points of a motion frame and sets motion to the selected joint using setJointFrames(). Also the model in the screen changes its pose upon setJointPose() method.

loadSelectedJointProperties() return the properties of the joint from the skeleton (min/max rotations on axes , number of axes available to set the proper motion.

getWord() loads the current word, in order to be update pictures and videos using the methods related with pictures, videos and categories. Calling setCategory() and getLabel() functions, current word can be edited.

Animation can be saved using saveAnimation().

Also the pictures, movies and current animation can be viewed using the methods provided by the base class.

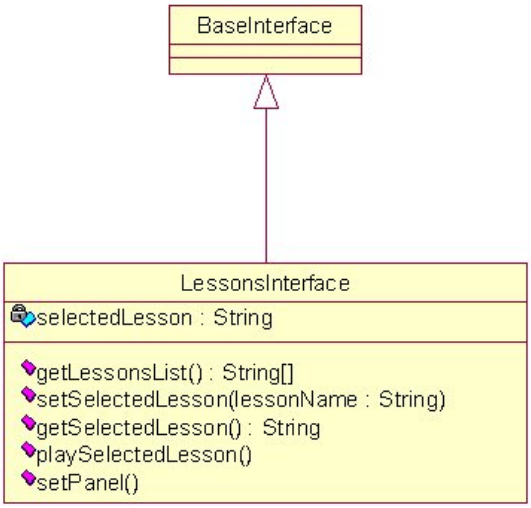

Figure 16: Lesson Interface

LessonInterface Class

This class [Figure 16] provides an interface to select and play flash animation lessons.

ExamEditorInterface Class

This class [Figure 17] provides the interface to prepare en exam. It contains an pointer to Exam object, all operations call methods of this Exam object., examName the string containing the name of the exam, a time integer an integer index to selected words and a char denoting the correct answer.

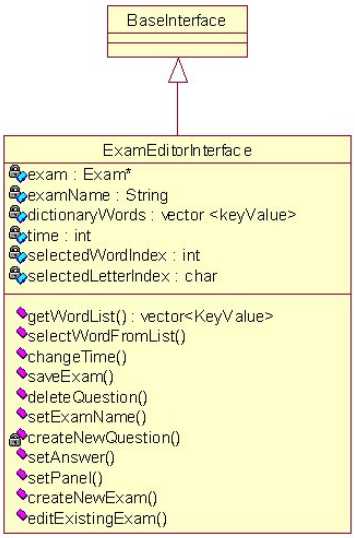

Figure 17: ExamEditor Interface

A new exam can be created using createNewExam(), loaded to be edited using editExistingExam(). A word can be retrieved according to its category. It can be added to the exam question as a selection. With setAnswer() function, a selection can be marked as true. A new question can be created or an existing question can be deleted. Using the methods createNewQuestion(), deleteQuestion() . changeTime() sets the exam time. At the end an exam can be saved.

TakeExamInterface Class

TakeExamInterface [Figure 18] provides user the interface to take an exam. It contains pointer to an exam object, a string vector holding the available exam names, string for the name of the exam, integers denoting the number of questions and score .

Exam list can be retrieved, an exam can be selected and started. Using the methods provided, user can answer a question, leave one unanswered and proceed to next questions.

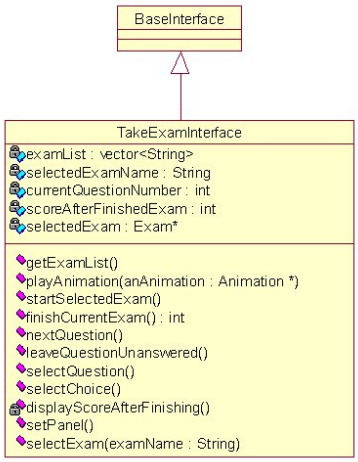

Figure 18: TakeExam Interface

When the exam is finished or the time is over, exam is finishCurrentExam() activated and the exam is evaluated.

4.2.2 MySkeleton and MyJoint Classes

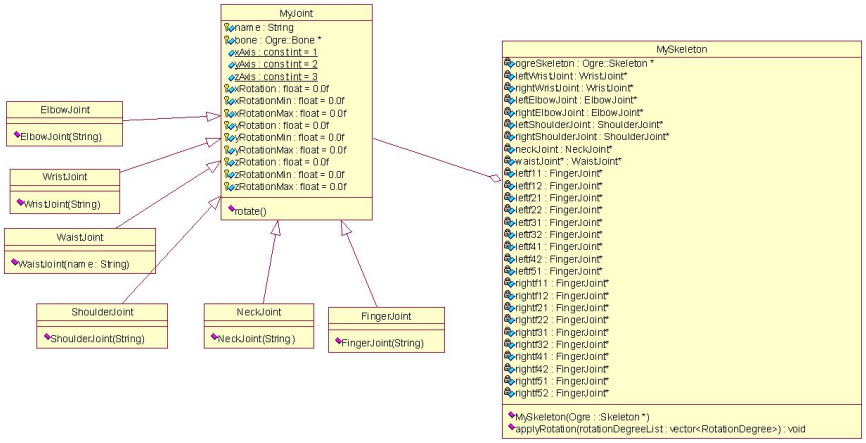

Figure 19: MySkeleton and MyJoint

MySkeleton class [Figure 19] is a class encapsulating the model which will be used in the animations. MySkeleton consists of a Ogre::Skeleton and numerous joints which are subject to move in the application.

MyJoint is an underlying class in MySkeleton. They consist of a pointer to an Ogre::Bone and float angle values which are the limits of rotation of joints . There are joint classes derived from MyJoint named as follows: WristJoint, ElbowJoint, ShoulderJoint, NeckJoint, WaistJoint, FingerJoint. These join types do not differ in methods but differ in rotation limits described above.

The joints are shown in the following skeletons [Figure 20 21] and the decision is made according to the procedure described below.

First of all, we start with selecting the initial model or its modified version from MakeHuman which is a tool, providing 3D virtual human. Exporting the file, with an .obj extension which is compatible with Blender3D, we can use the concepts that Blender3D supports. Besides these, we need to implant bones into the character for a realistic animation. Having the character, we should first build a hierarchy of bones that is namely the skeleton. Any place that can move, has a bone attached to it. The term Bone is used for specifying the rotation axis. For example, you can implant bones into the eyebrows of the character for providing eyebrow mimics even there is no such bone in real eyebrows. We need both forward and inverse kinematics concepts in order to keep the skeleton consistent and reasonable. Blender3D ensures forward kinematics if the hierarchy is correctly expressed. Having the parent child relation in hand, we can force the skeleton to support inverse kinematics. Inserting the required dummy bones to the skeleton, inverse kinematics can be implemented. Assuming a thoroughly matching skeleton object pair is acquired, we shall pass these to the Blender Ogre Exporter. This tool can get a Blender3D file and pass it to Ogre3D, besides splitting the file into the mesh file and the skeleton file. In Ogre3D, you can manually rotate and translate the selected joint.

4.2.3 Animation and Motion Classes

Animations [Figure 22] are stored in the instances of a class Animation, each animation consist of a motion list vector for each joint.

Motion class consists of float values for rotation angles for each axis, a start and finish time for the rotation.

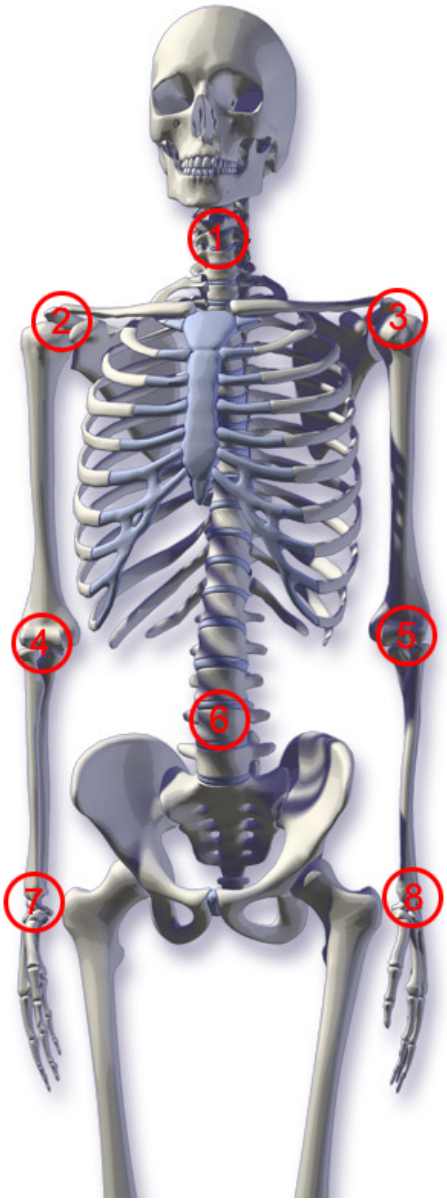

Figure 20: Skeleton

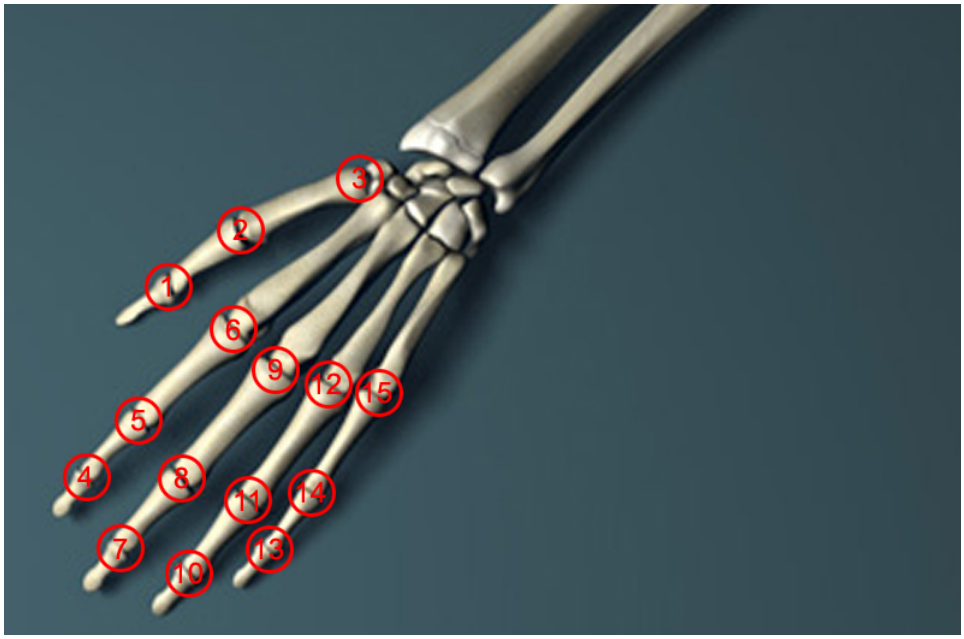

Figure 21: Hand

4.2.4 DataStoreQueryHandler Class

This is an operational class [Figure 23] connects PAPAĞAN with the datastore (xml files and database). Its methods provide the capability to retrieve animation and vocabulary lessons and exams from xml files, update and query the database, returning info and animation to the system.

4.2.5 AnimationPlayer Class

This is the class [Figure 24] that each interface class contain a pointer to one of its instances. As seen in the name it's a class that controls animation playing. playAnimaton() and playFingerspell() methods start animation and fingerspell. These methods are firing threads to play the related material. The playing threads (either single animation or fingerspell) can be stopped by stopAnimation() . rewindAnimation() sets the animation to a desired point.

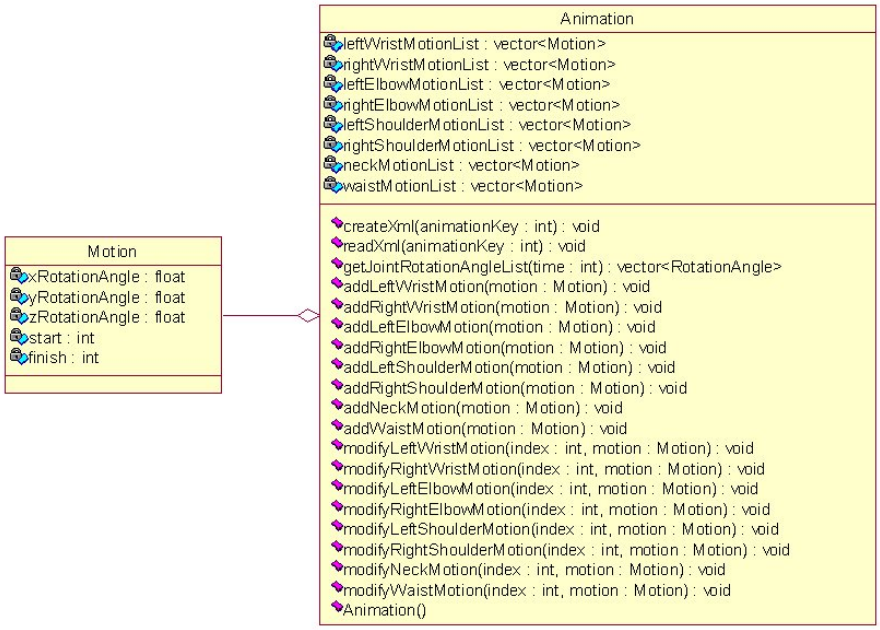

Figure 22: Animation and Motion

4.2.6 Other Classes

The other classes [Figure 25, 26, 27] are classes performing operations about editor, lessons, vocabulary lessons, game and exams. The main functionality and interaction is described in interface classes and their diagrams are illustrated below.

4.3 Interaction Diagrams

Dictionary

After setting DictionaryInterface [Figure 28], whole dictionary is retrieved from database via DataStoreQueryHandler object. As described in Dictionary Use Case Diagram in requirement analysis report, at any time, user searches for a word and plays either its sign animation or finger spell.

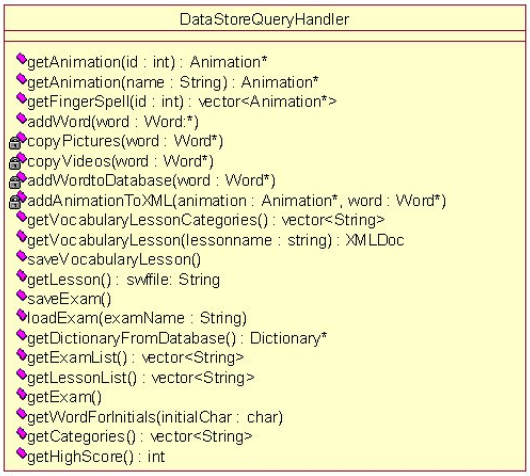

Figure 23: DataStoreQueryHandler

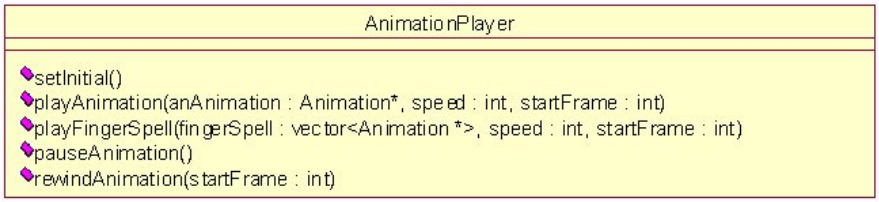

Figure 24: AnimationPlayer

Animation Editor

User has two basic choices [Figure 29], either editing an existing animation or creating a new one. However, for both of the choices, system allows to select a specific joint, set a time interval and modify the final position of the model for the selected joint. Obviously, user can save the animation as he completes setting model's pose for each time frame.

Exam Editor

After setting the Exam Editor Interface [Figure 30], system creates a blank exam then allows user to set it properties as duration and add any number

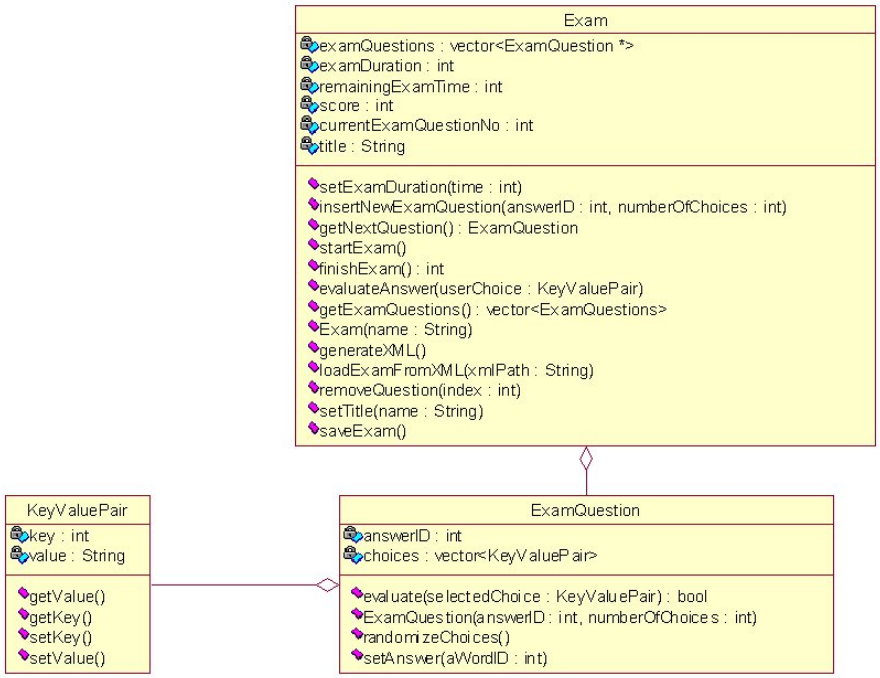

Figure 25: Exam

of questions. Obviously, exam is saved for further usage.

Exam

After Take 31 is set, all the exams are retrieved from data store in order to let user to choose one of them. When the time is up or user explicitly finishes the exam, score is displayed.

Games

As it is described in Initials Game [Figure 32] Use Case Diagram, in requirement analysis report, until user quits the game he receives new words as he succeeds in the previous one. When a wrong guess is made, user loses points. On the other hand, for a successful guess new points are gained.

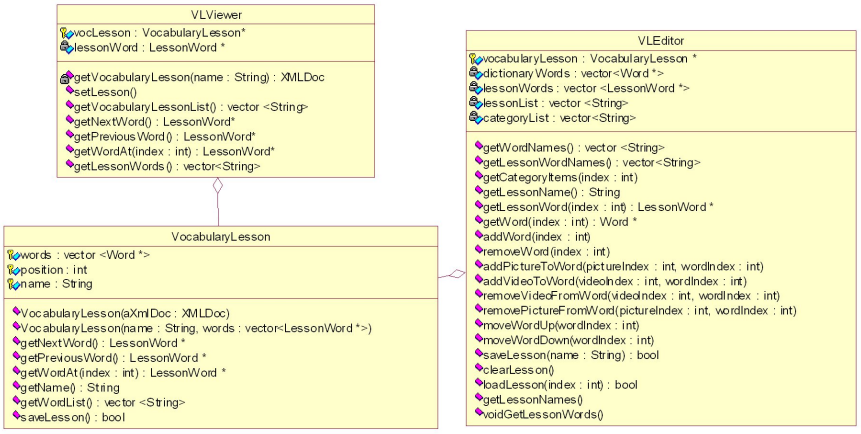

Figure 26: Vocabulary Lesson

Figure 27: Initials

Lessons

Most of the work is done by an external entity, in other words Flash Player, for the lessons section of PAPAĞAN and user only [Fgure 33]selects a lesson from Lessons Interface and plays it.

Vocabulary Lesson Editor

User either edits an existing lesson or creates a new one [Figure 34]. When he chooses to edit, the selected lesson, which is stored as an XML document,

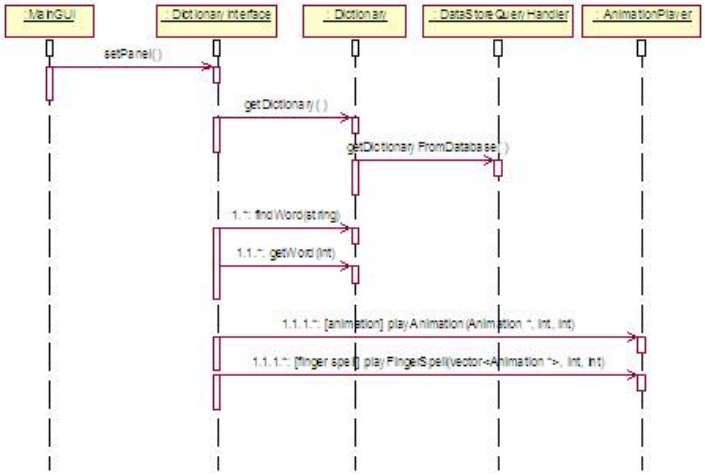

Figure 28: Dictionary

is retrieved from data store and loaded. For both of the cases, user shall insert/remove words, pictures and videos into/from the lesson. Obviously, changes are saved when user finishes his job.

Vocabulary Lesson

User simply selects an existing lesson [Figure 35] in the data store and starts it.

4.4 Data Design

Revising the ER diagram in the requirement analysis phase, word data will be kept in a mySQL database, vocabulary lessons and animation data will be stored in XML files whose DTDs are given below.

4.4.1 Database Tables

```
ITEM (
        ITEMID: INTEGER,
        ITEMNAME: CHAR(30),
        ANIMATIONPATH: VARCHAR(100)
  )
```

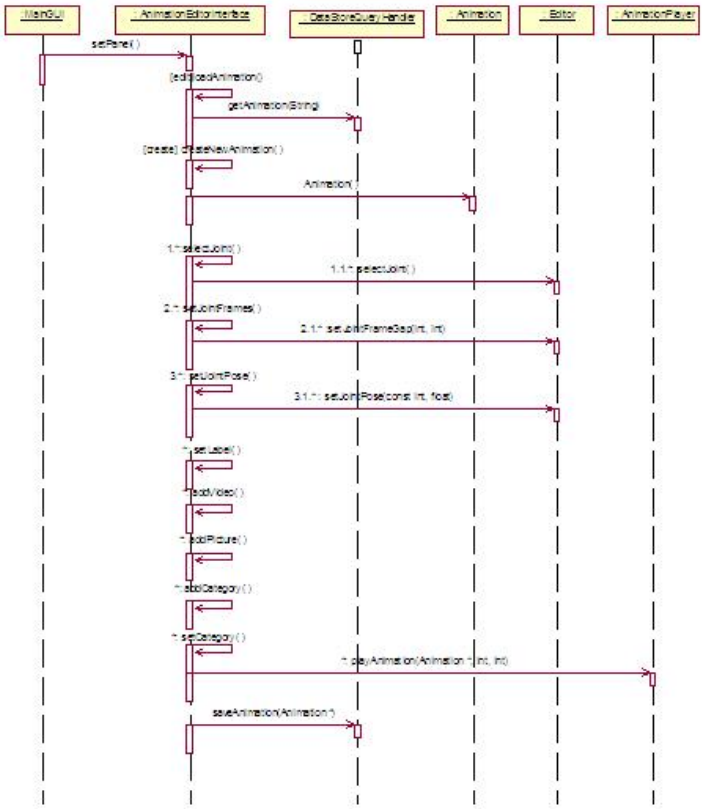

Figure 29: Animation Editor

PICTURE (
ITEMID: INTEGER,
PICTUREPATH: VARCHAR(100)
)

VIDEO (
ITEMID: INTEGER,
VIDEOPATH: VARCHAR(100)
)

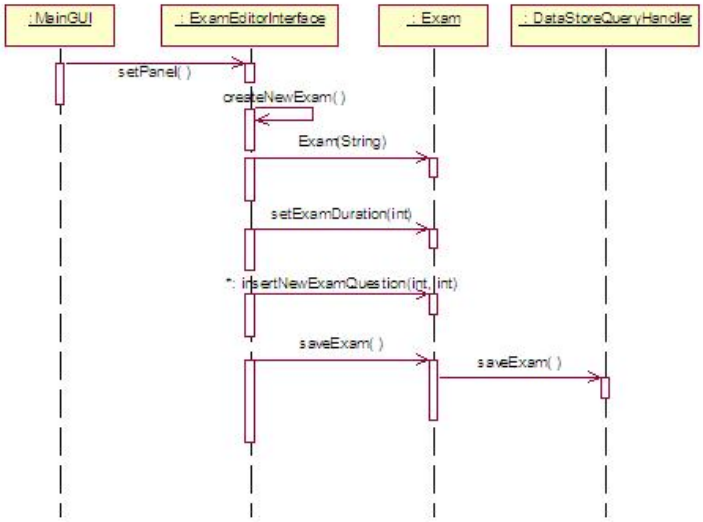

Figure 30: Exam Editor

CATEGORY (
CATEGORYID: INTEGER,
CATEGORYNAME: CHAR(100)
)

CATEGORIZED (
ITEMID: INTEGER
CATEGORYID: INTEGER
)

4.4.2 XML DTDs

- *Animation*

```
<!ELEMENT animation motion+ >
<!ELEMENT motion EMPTY >
<!ATTLIST motion joint (leftWristJoint | rightWristJoint |
      leftElbowJoint | rightElbowJoint | leftShoulderJoint |
      rightShoulderJoint | neckJoint | waistJoint | left11 |
```

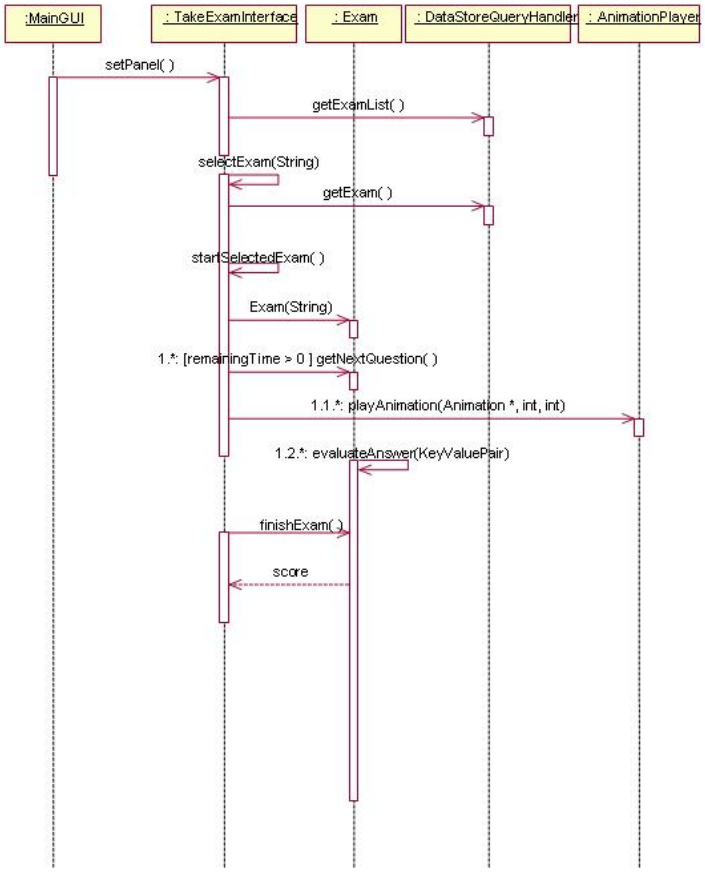

Figure 31: Exam

```
        left12 | left21 | left22 | left31 | left32 | left41 |
        left42 | left51 | left52 | right11 | right12 |
        right21 | right22 | right31 | right32 | right41 |
        right42 | right51 | right52)   #REQUIRED >
<!ATTLIST motion startingFrame CDATA #REQUIRED>
<!ATTLIST motion endingFrame CDATA #REQUIRED>
<!ATTLIST motion rotationAngleX CDATA #REQUIRED>
<!ATTLIST motion rotationAngleY CDATA #REQUIRED>
<!ATTLIST motion rotationAngleZ CDATA #REQUIRED>
```

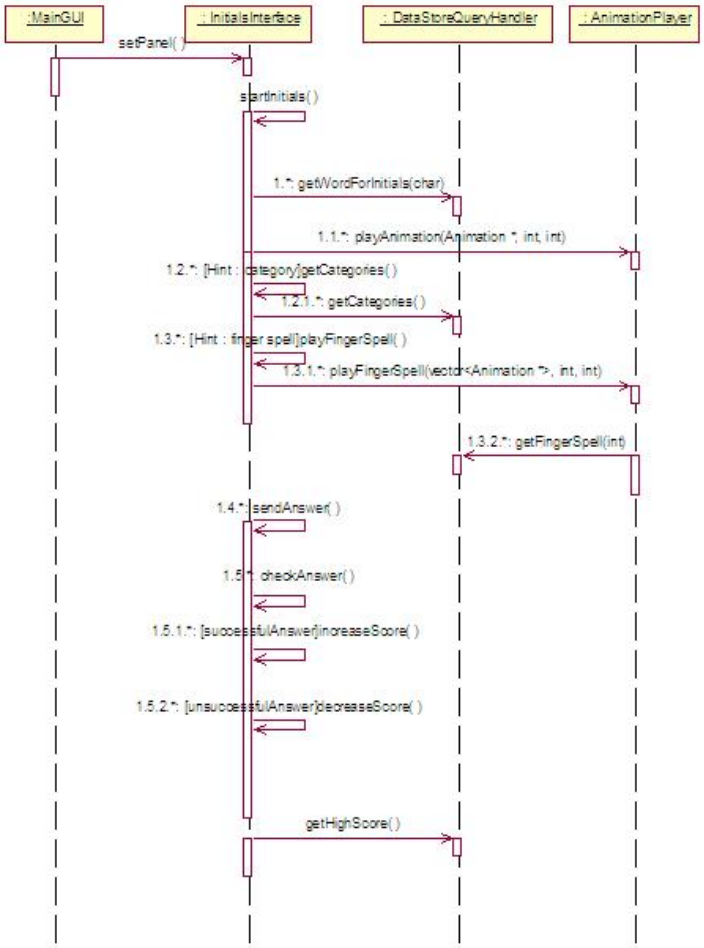

Figure 32: Games

- *Lesson*

```
<!ELEMENT lesson (word+)>
<!ATTLIST lesson title CDATA #REQUIRED>
<!ELEMENT word (animation, picture*, video*)>
<!ATTLIST word name CDATA #REQUIRED>
```

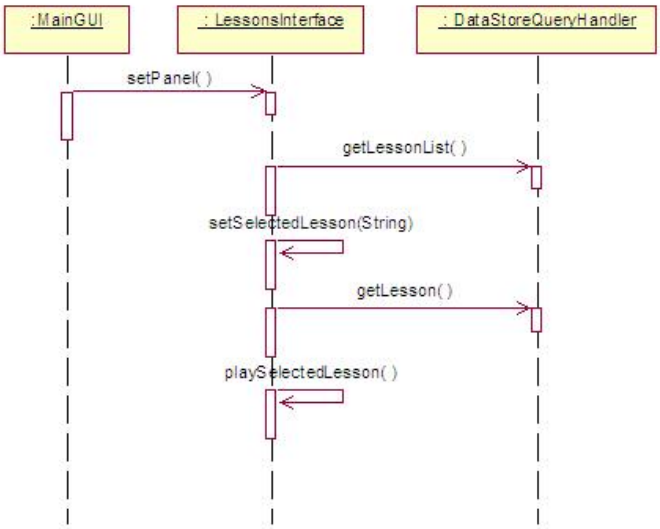

Figure 33: Lessons

```
<!ELEMENT animation (motion+)>
<!ELEMENT motion EMPTY>
<!ATTLIST motion joint (leftWristJoint | rightWristJoint |
       leftElbowJoint | rightElbowJoint | leftShoulderJoint |
       rightShoulderJoint | neckJoint | waistJoint | left11 |
       left12 | left21 | left22 | left31 | left32 | left41 |
       left42 | left51 | left52 | right11 | right12 |
       right21 | right22 | right31 | right32 | right41 |
       right42 | right51 | right52)   #REQUIRED>
<!ATTLIST motion startingFrame CDATA #REQUIRED>
<!ATTLIST motion endingFrame CDATA #REQUIRED>
<!ATTLIST motion rotationAngleX CDATA #REQUIRED>
<!ATTLIST motion rotationAngleY CDATA #REQUIRED>
<!ATTLIST motion rotationAngleZ CDATA #REQUIRED>
<!ELEMENT picture (#PCDATA)>
<!ELEMENT video (#PCDATA)>
```

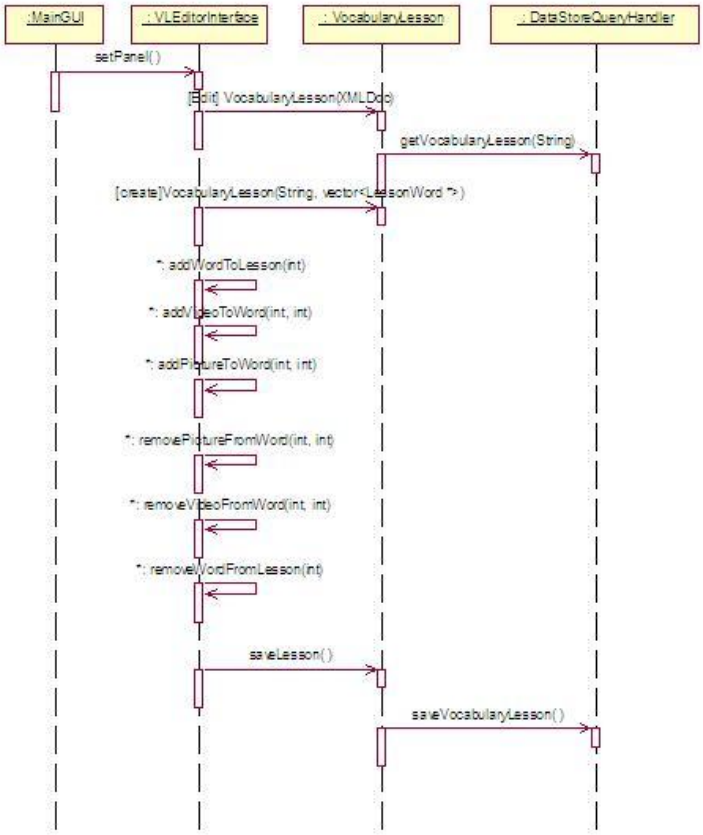

Figure 34: Vocabulary Lesson Editor

- *Exam*

```
<!ELEMENT exam (question+)>
<!ATTLIST exam title CDATA #REQUIRED>
<!ATTLIST exam duration CDATA #IMPLIED>
<!ELEMENT question (choice+)>
<!ATTLIST question answerID CDATA #REQUIRED>
<!ELEMENT choice EMPTY>
<!ATTLIST choice id CDATA #REQUIRED>
```

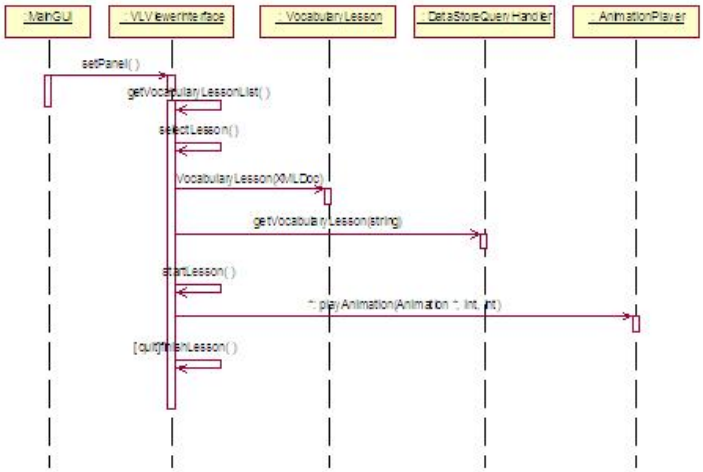

Figure 35: Vocabulary Lesson

References

[1] http://www.techiwarehouse.com/cms/engine.php?page_id=18a41ffa

[2] http://cs.wwc.edu/KU/SEBOOK/Design.html

[3] Booch, Rumbaugh, Jacobson. The Unified Modeling Language User Guide Addison Wesley, 1998