Design Patterns for Multimedia Mobile Application

Manal Mostafa Ali, A.Fattah Elsharkawi, M. G. El- Said, M. Zaki

Computer & System Department, Al-Azhar University, Cairo – Egypt

Manalmustafa10@yahoo.com, sharkawi_eg@yahoo.com, mgamal@4s-systems.com, mzaki.azhar@gmail.com

Abstract

This paper presents a Multimedia Mobile Framework, MMF, to provide mobile applications with flexibility and reusability. The proposed framework integrates fundamental design patterns (GoF), visualization patterns, and sound patterns. The fundamental design patterns include, factory method, composite, state, and flyweight after adapting them for Android environment. The visualization patterns are modified and adapted to match MMF. Eventually, the sound patterns to convert from text to speech and vice versa are introduced as new patterns for mobile applications. In all stages, the Android Activity class is employed throughout MMF patterns.

To emphasize the significance of MMF, it has been relied upon to develop a multimedia editor for M-Learning applications running on Android platform. In addition, specific quality metrics were selected and the performance of such pattern based editor is quantitatively compared with that of traditionally developed one.

Keywords: *Design patterns, Information visualization, Sound patterns, Android, Multimedia editor, Quality metrics.*

1. Introduction

Design patterns [2] [9] [33] are gaining popularity because they support modifiability and flexibility of designs [8] [35] [37]. They are effective design methods which when used carefully in the implementation of software development will lead to several advantages. A pattern in software community has been defined by Gamma et.al. [14] as a description of communicating objects and classes that are customized to solve a general design problem in a particular context. Reusability, encapsulation [5] and ease of system maintenance are some of the benefits one can incorporate into software systems if these proven design patterns are used in the system [41].

Previous researches have used software design patterns to develop different applications on mobile platforms. Such applications include, proposing an event_based finite state machine to facilitate mobile input handling [42]. In that work, the authors have proposed an approach to develop an input method for mobile using the state design pattern and has examined it to develop a third party input method system which contains two versions: a Windows mobile system version and a Symbian system version. Ali et. al. [3] and Bian et. al. [7] discuss a methodology developed for designing software in the context of frameworks for showing how design patterns can serve as the bridge between the paradigms imposed by the framework and the ideal, unconstrained design of the system. However, all these applications lack any common mobile framework.

In this paper, MMF is presented as a common multimedia mobile framework that can be used for Android [11] [15] [27] [30] environment. It consists of:

1. Fundamental (GoF) patterns [14],
2. Modified visualization pattern(s)[18], and
3. Sound patterns.

Accordingly, the contribution of this work is:

1. Introducing two new sound patterns, one to synthesis text to speech, and the other to convert from speech to text supporting realistic distribution multimedia framework offering a multitude of features to the users.
2. Extending the reference model pattern [18] to provide a generic interface for both structuring visualization and listening in a flexible and reusable fashion.
3. Customizing design patterns to include Android Activities and Views as distributed classes through MMF patterns.
4. Integrating a layered architecture of software patterns in a semantic network that establishes the inter-relationships between the underlying patterns.

To examine the applicability of MMF, it has been used to develop a multimedia editor for M-Learning applications. Consequently, the performance of such editor is compared with that of a traditionally developed one. The comparison as such is based on CK [36] metrics and other traditional metrics [5] [26].

The rest of this paper is organized as follows: Section 2 introduces related work. In section 3, we analyze design patterns used in MMF. The corresponding semantic network is given in section 4. Section 5 discusses the merits of using MMF patterns and gives a quantitative comparison between MMF and a traditional approach. Section 6 concludes the paper.

2. Related Work

Plakalović and Simić [31] have considered the usability of MVC (Model-View-Controller) and PAC (Presentation-Abstraction-Control) design patterns for the separation of the user interface tasks from the business logic in mobile application. They define the problem of differentiating MVC from PAC software patterns when designing mobile applications. Tomáš [40] is an author that highlights the possibility of using design patterns in mobile architectures under Android environment. He describes a basic example of using factory method in Android, and exploited the corresponding API named SSLSocketFactory to create specific SSLSockets. Similarly, Grace [16] discusses the usage of design patterns in a number of applications one of them is devoted to Android environment. He points out that Android is a pattern based OS. Examples of Android APIs are: 1) the MediaPlayerService class in which the factory method has been exploited to create different types of concrete media players, 2) Activity class that uses the template method pattern to manage interactively the user actions, and 3) Intent class which use the command pattern.

Hanafi, Samsudin [17] and Shanmugapriya, Tamilarasia [38] have developed an interactive M- learning application based on Web services using Android mobile devices to facilitate the ubiquitous learning. The application developed by [38] consists of three

activities: i) Module selection, ii) Course selection, and iii) Quiz question and answer choices. All the available courses, tests, and assignments are loaded from a database that has been stored on the server side. A dedicated Web service method is used for this purpose.

Here, we focus on design patterns as a basic orientation of MMF to develop an interactive editor prototype that can be used in M-Learning applications.

3. Design Patterns for MMF

Following the notation of Gamma et. al. [14] each pattern of MMF consists of a name, intent, and a summary description. In addition, each pattern is illustrated by a structural diagram to depict the participants, classes and the relations between them, such as inheritance, reference, and aggregation. Moreover, Android activities are exploited as distributed classes throughout MMF.

It is worth mentioning that an activity is the presentation element of every application [19]. Usually, every activity coincides with a single screen [6]. In our work, MyActivity is implemented as a single class that extends the Activity class and displays the user interface. Navigating from screen to screen is accomplished by resolving Intents. All activities need to be declared in AndroidManifest.xml file (with precisely that name) in its root directory [25].

3.1 Adapted GoF design patterns

The underlying GoF design patterns, after adapting them for Android include: A_factory method, A_composite, A_state, and A_flyweight pattern.

3.1.1 A_Factory Method pattern

Intent: The A_factory method provides a simple decision making class that returns one of several possible subclasses of an abstract class depending on the data that are provided [10]. It is commonly used when defining frameworks because they do not know and should not be concerned with instantiating specific objects. The class diagram of the A_factory method after modifying it to cope with Android is shown in

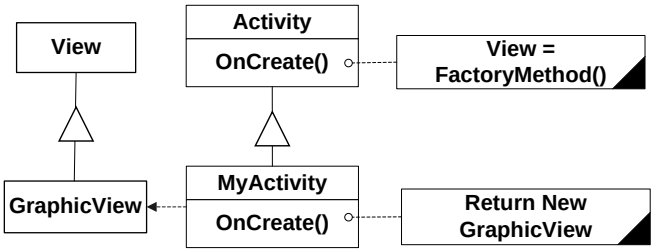

Figure 1- A_Factory method pattern

.

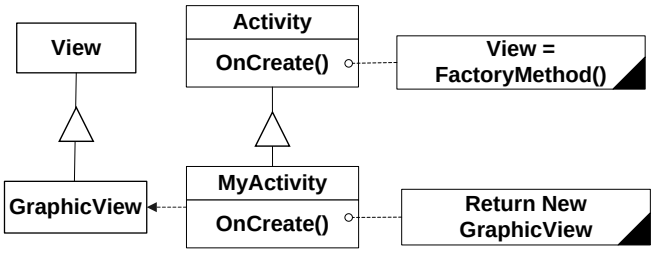

Figure 1- A_Factory method pattern

Participants- the participants of that pattern are:

- Activity:1) declares the factory method and returns an object of type View, 2) defines a default implementation of the factory method that returns a default GraphicView object, and 3) may call the factory method to create a View object.
- MyActivity: overrides the factory method to return an instance of a GraphicView.
- View: has a factory method that returns an object that conforms to the interface defined by that class. Typically, a class which implements A_factory method pattern might be an abstract class with several concrete subclasses [12].
- GraphicView: as factory methods eliminate the need to bind MMF specific-classes into the code, then, the user of an underlying application will be able to deal only with the View interface via that GraphicView classes which implement the View interface.

3.1.2 A_Composite pattern

Intent: Improving code maintainability through the avoidance of code duplication and the reuse of modular UI components is described in the A_composite pattern. The pattern allows a group of objects to be treated in the same way as a single instance of an object. The selection of this pattern is justified by its usefulness for decomposing a complex object into a hierarchy of components, in which all the constituting components would be treated in a similar manner [28]. Furthermore, Android UIs are defined as a hierarchy of View and ViewGroup nodes. The draw() method will draw the individual widget or the composite widget [16]. The structure of A_composite pattern is indicated in

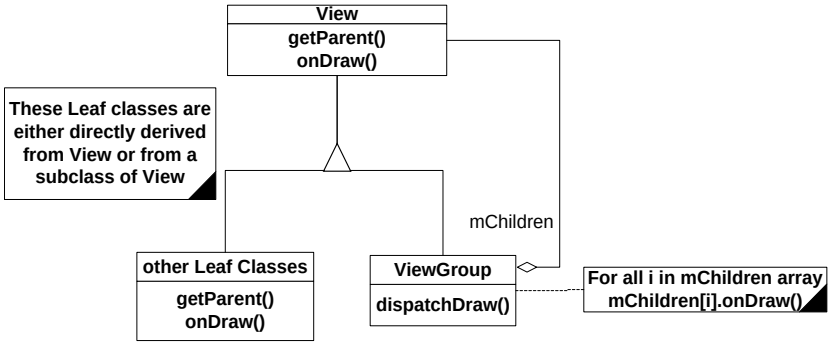

Figure 2.

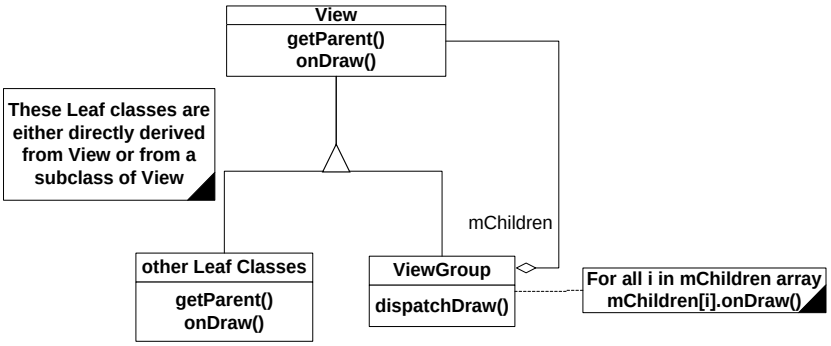

Figure 2- A_Composite pattern

Participants- the participants of that pattern are:

- View: The View class is the Android's most basic component from which graphical users interfaces can be created. All the UI elements and some invisible items are derived from this class such as Textview, Button, Edittext, Scroll Bars and so on. This element is similar to the Swing JComponent class for Java apps.
- ViewGroup is a special view that can contain other views (called children). It is the base class for layouts and views containers [4].
- Leaf represents leaf objects in the composition, has no children, and defines behavior for primitive objects in the composition.

3.1.3 A_State pattern

Intent: The focus is to make a screen that appears to the students with at least two states: animate mode and stop mode. A_State pattern is used for this purpose. It allows an object to alter its behavior when its internal state changes. The object will appear to change its class [7]. The structure of A_state pattern is illustrated in Figure 3 – A_State pattern.

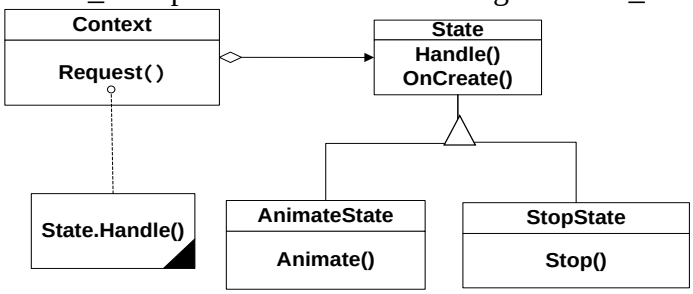

Figure 3 – A_State pattern

Participants- the participants of that pattern are:

- Context: the Context is the role of the class that changes its behavior. This is achieved by encapsulating the behavior in several objects, each defining a state of the Context and only one being active at a time. The Context delegates all the received requests to the current state object.
- State is the interface to the internal state of the Context. The state interface exposes a set of operations common to all states and is implemented by concrete state objects.

- Concrete states (animate and stop) in MMF are concrete implementations of the state interface, providing the behavior of specific states. The Context class has a reference to a state instance, to which it forwards all behavior related requests.
-

3.1.4 A_Flyweight pattern

Intent: Designing objects down to the lowest level of system "granularity" provides optimal flexibility, but can it be unacceptably expensive in terms of performance and memory usage. A_flyweight shows how memory occupation can be minimized by sharing as much data as possible between similar objects. The structure of the A_flyweight is indicated in Figure 4.

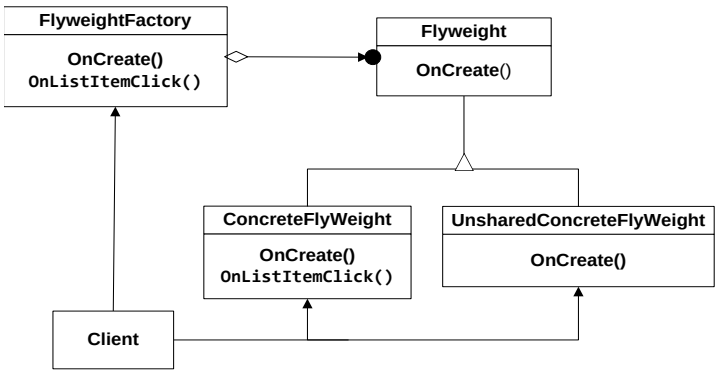

Figure 4 – A_Flyweight design pattern

Participants- the participants of that pattern are:

- Flyweight: is the interface implemented by objects sharing state. The operations defined in the Flyweight interface accept the shared state (extrinsic state) as parameters.
- FlyweightFactory: is a class responsible with the creation and management of flyweight instances.
- ConcreteFlyweight: implements the Flyweight interface. It 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. The Flyweight interface enables sharing; it doesn't enforce it.

3.2 Adapted information visualization design patterns

Two patterns were selected from the domain of information visualization to be included in MMF in order to enhance its capabilities; namely integrated reference model pattern and A_scheduler pattern.

3.2.1 Integrated Reference Model pattern

Intent: In order to be reusable, easy to add, and remove, the components implementing transformational operations should be independent and loosely coupled with the other components of the system. Reference model pattern addresses these issues. Moreover, this

pattern is extended to provide integrated interface from both multiple visual and listening representation of the model, supporting multimodal framework that enable the users to interacts with the editor through editing, reading, or listening. The structure of integrated reference model pattern is shown in

Figure 5.

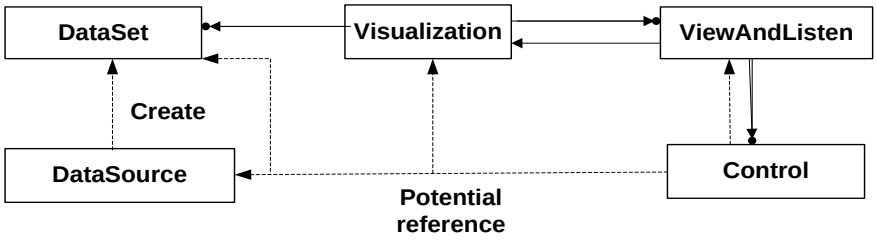

Figure 5: Integrated Reference Model Pattern.

Participants- the participants of that pattern are:

- Visualization: manages visual models for one or more data sets, separating visual attributes (location, size, color, geometry, etc) from the abstract data.
- ViewAndListen is a visual representation of the model and is capable of showing one or more representations of that model. One or more views provide a graphical display of the visualization.
- Control: modules process user input and may trigger updates at any level of the system. Moreover, the data access is better controlled because it's also separated from GUI and processing. So the same data entered by the user can have several visual representations on the editor. The Visualization, ViewAndListen, and Control classes employ the standard Model–View-Controller (MVC) [23] pattern of user interface development to separation the user interface tasks from the functional core. The View manages the display of information, and interacts with the user through elements of the GUI. Moreover, we extend the View to become ViewAndListen allowing the editor to be multimedia offering a way of making application more comfortable for the special case users.
- A DataSource is a component such as a formatted file reader or a database connectivity interface. For Android, we use a shared preference to loads data sets to be visualized. SharedPreferences is a class provided by the Android framework and provides storage for key/value pairs of primitive data types [32].
- DataSet: one or more data sets can then be registered with visualization. This mechanism allows abstract data to be separated from visual attributes such as location, size, shape, and color, thereby allowing a single abstract data set to be used in multiple visualizations.

3.2.2 A_Scheduler pattern

Intent: Appending animation to the application requires updating the display over specified durations. Graphical and text animation were supported by examining A_scheduler

pattern. Animation can be a very effective mechanism to convey information in visualization and user interface settings [20]. It requires updating visual properties and redrawing the display at regular time intervals, usually over a specified duration. A_Scheduler pattern was employed to show how flexible, robust, and reusable support for animation can be incorporated into Android user interface. The structure of the A_scheduler pattern is shown in Figure .

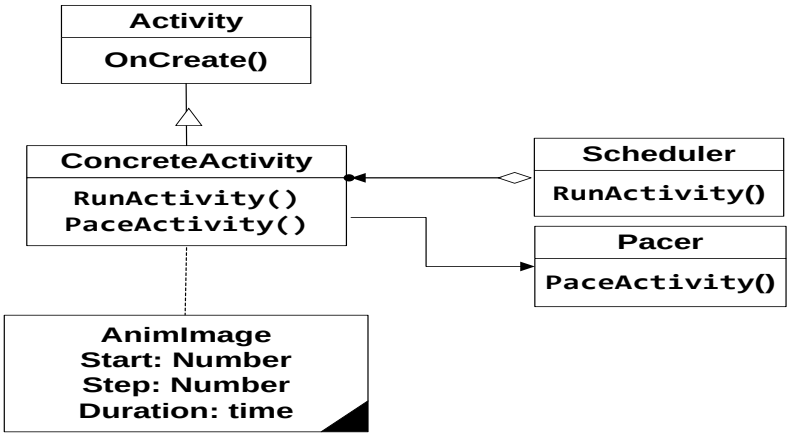

Figure 6 – A_Scheduler pattern

Participants- the participants of that pattern are:

- Activity: Android activities are used as distributed components along with the application. An activity is single and focused thing that a user can do.
- ConcreteActivity: Custom operations are created by subclassing the Activity class. Every new activity must be named in manifest. Name is used as a parameter that is passed to the constructor of intent which is used to start desired activity. The developer defines the layout of the activity which the activity displays to the user. An activity has a specified start time, duration, and a step time defining the desired length of time to wait between repeated invocations.

For instance, each object which wishes to respond to animation requests simply implements the following three methods: start_transition, transition_step, and end_transition.

The start_transition message informs the object that it is about to begin a new transition. A series of transition_step messages then indicate how the object is to advance along the transition over time. Finally, an end_transition message is passed at the final step indicating that the transition has completed. Each transition_step and end_transition message contains parameters that indicate both the start and end positions of the animation step being taken [20].

- Scheduler: Activity instances are registered with a centralized scheduler that runs the activities over a specified time interval, repeatedly running the activity at requested time steps.

3.3 Sound patterns

The key additional patterns of MMF are the sound patterns. These patterns are:

i) A_Text To Speech, A_TTS, is designed and implemented to transform text to speech.
ii) A_Speech To Text, A_SST, is constructed to convert speech to text.

3.3.1 A_Text to Speech pattern

Intent : Reading or understanding a text can at times be a challenge if it is written in a foreign language, if the reader is illiterate, or if one has reading disabilities. A _text to speech (A_TTS) pattern is a system that can read text aloud automatically. A_TTS system converts normal language text into speech. A synthesizer can incorporate a model of the vocal tract and other human voice characteristics to create a completely synthetic voice output [29]. By using this pattern we can develop MMF that can read a text for the users through mobile Android devices. The following list presents cases in which the pattern can be used:

- Blind people can use it when they have a text to read [1].
- Non-native learners can use it when they do not understand a text written in a foreign language, or when they are not sure of the right pronunciation of words.
- The illiterate or dyslexic can use it when they have difficulties reading a text. The structure of A_TTS pattern is shown in Figure 7.

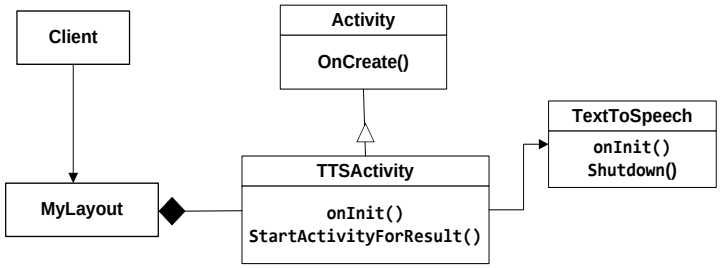

Figure 7: A_Text to speech pattern

Participants- the participants of that pattern are:

- Activity: the most basic building block of an Android application [13].
- TTSActivity: extends the Activity class and is responsible for getting text entered by users and initializing TextToSpeech Android API.
- TextToSpeech synthesizes speech from text for immediate playback or to create a sound file [4]. A TextToSpeech instance can only be used to synthesize text once it has completed its initialization, implement the TextToSpeech.OnInitListener to be notified of the completion of the initialization and provide the means to turn text into audio files or play back the result directly.
- Figure 8 is a means to summarize text to speech conversion stages.

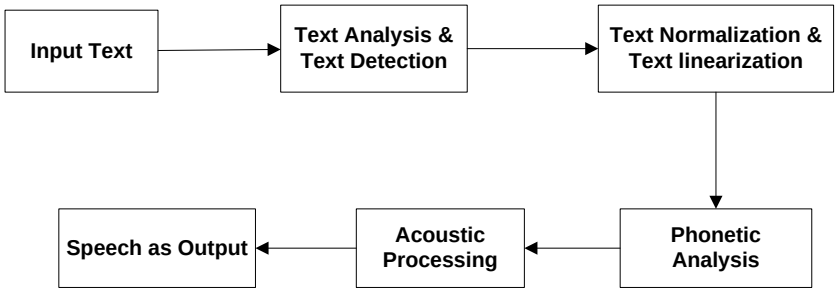

Figure 8: Text to speech conversion [29]

- MyLayout is the A_TTS view that manages the display of information, and interacts with the user through elements of the GUI.

Moreover, Android also has an option where the voice can be changed from the default voice, allowing a device to synthesis with any accent. The TextToSpeech synthesis relies on a dedicated service that runs across all apps that use synthesis. So, when the application terminates, it should call the shutdown () function from the TextToSpeech API to release the native resources used by the TextToSpeech engine. A_TTS can be used to read text displayed on a regular computer and also to read messages on handheld devices such as SMS. This tool is invaluable for users with visual impairments and it is also in GPS devices providing directions and addresses to drivers. This system provides a user friendly interface easy to handle for blind users: the buttons to access the different functionalities of the software in the editor are located at the corners of the screen and are therefore, easy to access.

3.3.2 A_Speech to Text pattern

Intent: Blind and deaf users suffering from interaction with such editor. We want to eliminate this burden and simplify the connection between them. This pattern allows you to simply speaking on your phone and let the phone do all the work translating your words into text. For instance, speech recognition is done via the Internet, connecting to Google's server [34]. Google's Speech recognition service records what the user has to say on the phone and offloads it's processing to the cloud; it then reflects accurately what the user said. It can recognize any length of speech and rare words even with an accent. The service uses artificial intelligence to recognize speech. Google stores millions of speech utterances from the users in order to improve their service. The following list presents cases in which the pattern can be used:

- Blind people, can’t edit on the screen but can communicate with MMF through speaking.

Deaf and mute users have listening difficulties, but can see what the instructor says

- after speech conversion. The structure of A_STT pattern is shown in
- Figure 9.

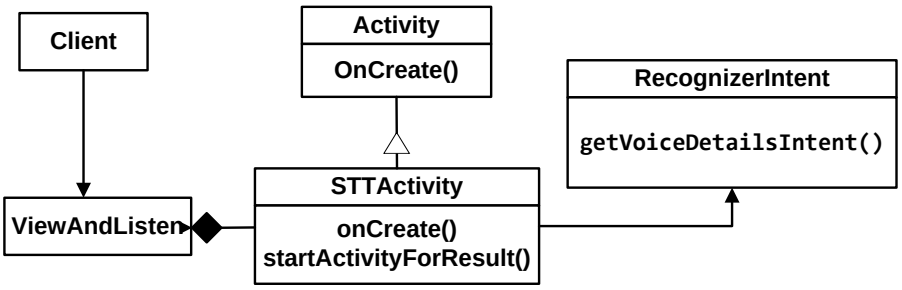

Figure 9: A_Speech to Text pattern

Participants- the participants of that pattern are:

- STTActivity is the startup activity defined as launcher in AndroidManifest.xml file. This class is responsible for rendering the display, sending text to speakers, generating tactile feedback, and so on. Starting an activity that will prompt the user for speech, send it through a speech recognizer, and either display a web search result or trigger another type of action based on the user's speech.

  REQUEST_SPEECH is static integer variable, declared on the beginning of activity and used to confirm response when engine for speech recognition is started. REQUEST_SPEECH has positive value. Results of recognition are saved in variable declared as ArrayList type. Method onCreate is called when activity is initiated. This is where the most initialization goes: setContentView (R.layout.viewandlisten) is used to inflate the user interface defined in res > layout > viewandlisten.xml, and findViewById(int) to programmatically interact with widgets in the user interface. This method has also a check whether mobile phone, on which application is installed, has speech recognition possibility.
- RecognizerIntent: Recognition process is done through one of Google's speech recognition applications. If recognition activity is present user can start the speech recognition by pressing on the button and thus launching startActivityForResult (Intent intent, int RESULT_SPEECH). The application uses startActivityForResult() to broadcast an intent that requests voice recognition, including an extra parameter that specifies one of two language models. Intent is defined with intent.putExtra (RecognizerIntent.EXTRA_LANGUAGE_MODEL, "en-US").
- ViewAndListen interacts with the user through elements of the GUI. ViewAndListen is responsible for receiving voice from client and sending to STTActivity. User actions on this class will trigger events that will go to the application functions. In order to run the speech recognition from Google one needs to make an "intent" object with the intent RecognizerIntent.ACTION_RECOGNIZE_SPEECH. startActivityForResult(), this then places a message on the message queue, and when the program pauses, Android will go through the messages and when it reaches that message it will start a new activity and return the result of the activity.

Since the start of MMF, displays on mobile phone show button which initiate voice recognition process. When speech is detected, the application opens connection with Google's server and starts to communicate with it by sending blocks of speech signal. Simultaneously the Figure of waveform is generated on the screen. Speech recognition of the received signal

is performed on server. Google has accumulated a very large database of words derived from the daily entries in the Google search engines as well as the digitalization of more than 10 million books in Google Book Search project. The database contains more than 230 billion words. If we use this kind of speech recognizer, it is very likely that our voice will be stored on Google's servers. This fact provides continuous increase of data used for training, thus improving accuracy of the system [34].

4. The Semantic Network

Figure 10 depicts the relationships between the underlying software patterns graphically. Moreover, such patterns are arranged in a layered model that has three layers. The lower layer includes the GoF patterns after adapting them for Android (A_factory method, A_composite, A_state, and A_flyweight), the middle layer has information visualization patterns after modifying them to mobile applications (integrated reference model, A_scheduler), and the highest layer includes the new sound patterns (A_text to speech and A_speech to text).

Examples that comprise several essential relationships are indicated as follows: A_flyweight is often combined with A_composite to implement shared leaf nodes, while combining the A_composite pattern with A_speech to text pattern can provide a support for powerful and flexible data representation. The integrated reference model uses A_scheduler pattern to animate view transitions, while combining the same pattern with A_factory method pattern results in widget and different objects creation. Actually,

Figure 10 provides a roadmap to applying MMF patterns. Furthermore, its significance might be illustrated by straightforward structuring an interactive editor prototype for M-Learning.

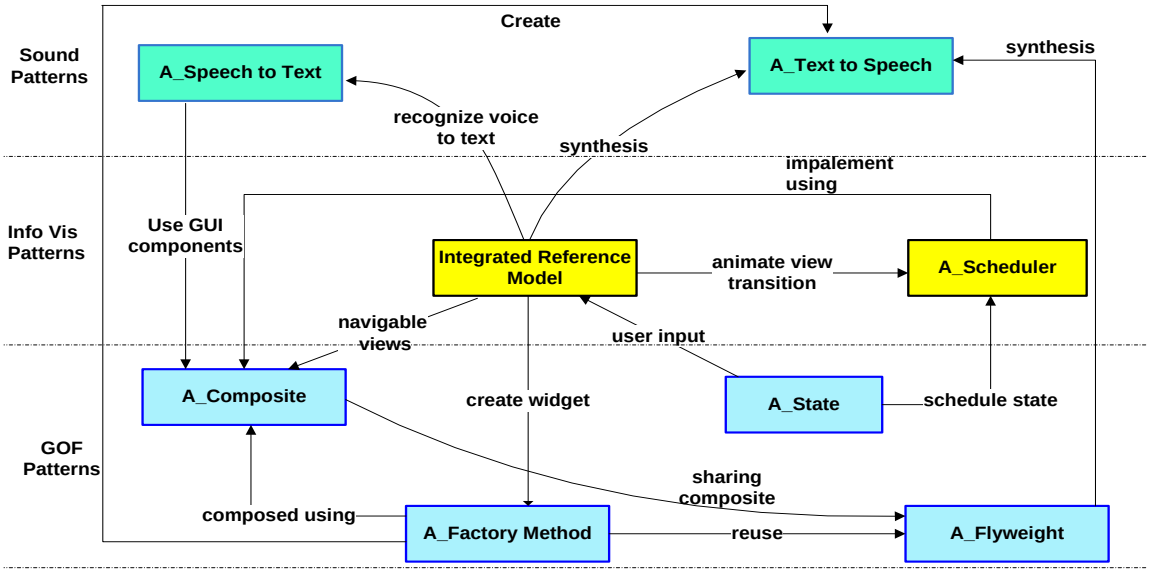

Figure 10: MMF semantic network

To accomplish such interactive editor, the A_factory method pattern is exploited to avoid creating objects directly. The A_factory method uses A_TTS to synthesize each word appears on the screen. Moreover, A_TTS enables the user to write his own words on the screen, where these words are converted directly into speech.

All GUI elements are created using the A_composite pattern. View class is the basis of a graphical user interface in Android. Android organizes its UI elements into layouts and views. Everything the user sees, such as a button, label, or text box, is a view. Layouts are responsible for organizing views [13]. Therefore, the A_composite pattern is related nearly with all patterns used in MMF. Also, the integrated reference model pattern is related to all MMF patterns. It makes use of loose coupling with A_SST pattern to convert voice to text and to examine the A_scheduler pattern. Actually, the A_scheduler pattern is responsible for text and image animations with regular durations. Activity instances can incrementally update the view transformation and trigger repaints.

The A_flyweight is a pattern for sharing objects, where each instance does not contain its own state, but stores it externally. This allows efficient sharing of objects that will lead to space saving. A_State pattern is used as an alternative to conditional logic. That is, the A_state pattern puts each branch of the conditional in a separate class. Thus, one can treat the object's state as an object in its own right which can vary independently from other objects.

5. The Merits of using MMF Patterns

- The A_factory method pattern is used to loosen the coupling between a class (Activity) and another class that it instantiates (View). Specifically, it enables the activity class to defer instantiation to a subclass; in this way it is easy to extend the activity class to work with a new type of View class. This technique is widely used among programmers to solve issues when creating instances on runtime, allow the creation of various instances of classes which implement the same interface, and can be then delivered to a class which operates above them using the interface they implement. The classes using the interface don't care which implementation we deliver to them. This allows the client to be able to continue production of objects for MMF without having to worry about specifying (or even knowing) its concrete type in a dynamic environment.
- The A_composite pattern allows an instance of a class to be treated in the same way as a group of objects. It makes it easy to add new kinds of objects. It makes clients simpler, because they do not have to know if they are dealing with a leaf or a composite object. Thus, its use in a system impacts positively the expandability and the simplicity, which is in accordance with the evaluations of our respondents.
- A_State pattern promotes high cohesion, polymorphism and protected variations principles stated by Bansiya, Davis [5] and Larman [24]. In addition, the code is easier to follow. The practical value is that having a complete state system in place is beneficial because it allows relevant input events to be presented more clearly and quickly to the users.
- The A_flyweight pattern is a means to allow clients to share as much common information as possible while allowing extrinsic information about objects to be

maintained. Thus, using A_flyweight in MMF reduces lots of memory consumption and minimizes unacceptable run-time overhead upon executing mobile application.

- The integrated reference model pattern provides a general template for structuring visualization applications that separates data models, visual models, views, and interactive controls. Actually, it decouples the presentation layer from the business logic as well as to change the presentation without changing the code. Also integrated reference model pattern determines the right separation of concerns which has essential consequences for the complexity, extensibility, and reusability of software architectures. Since the integrated reference model pattern provides a loose coupling between components, therefore, each component can easily be reused, modified or replaced.
- By using A_scheduler pattern, it is possible to integrate animated presentation with more conventional user interface techniques. It promotes a solution that enables time sensitive operations and supports extensibility. It has been used extensively in visualization and user interface frameworks.
- Successful use of A_TTS pattern isolates TextToSpeechEngin from user interface considerations, resulting in an application which is easy to modify. Actually, either the visual appearance or the underlying business rules could be modified without affecting each other. In addition, the A_TTS pattern aims to help people with reading disabilities, or illiterates and non-native speakers to hear the content of text they have difficulties to read.
- By making use of A_SST pattern, we can improve system accessibility by providing data entry options for blind, deaf, or physically handicapped users [34]. Consequently, an interactive editor with multi-modal interfaces supporting (text, image, and audio data formats) can be generated.

In addition to these qualitative metrics, a quantitative quality indication might be obtained by comparing the quality metrics of the design patterns approach with that of the traditional object oriented approach when both approaches are used to design a multimedia mobile editor. Accordingly, specific software metrics are applied on both MMF and corresponding object oriented design. These metrics include CK [36] and a traditional metrics [5] [26]. Such Parameters are automatically computed by Eclipse plug-ins [Metrics 1.3.6, http://metrics.sourceforge.net/].

From CK metrics we focus mainly on two metrics:

- Weighted method per class (WMC) [21] [22] as a measure for reusability.
- Lack of Cohesion of Methods (LCOM-CK) (as originally proposed by Chidamber & Kemerer) describes the lack of cohesion among the methods of a class [26] [39].

While traditional metrics are concerned with:

- Lines Of Code (LOC) [26].
- Number Of Classes (NOC) [5].
- Number Of Methods (NOM) One that simply counts the number of methods in the class [26]. The results of the comparative study are listed in Table 1.
-

Error! Reference source not found.- Comparison of MMF and a traditional development approach

	Metrics						
	WMC		LCOM		Traditional metrics		
Project	Mean	Std. Dev.	Mean	Std. Dev.	LOC	NOC	NOM
OOP	4.409	6.548	0.187	0.271	857	22	26
MMF	3.667	5.38	0.102	0.212	1006	33	58

From**Error! Reference source not found.**, one can point out the following:

- MMF decreases the value of WMC i.e. reduces the class complexity. The larger the number of methods, the greater the potential impact on children classes, as they inherit all methods of parent classes [39]. Also, the number of methods and the complexity of the methods involved is an indication of how much time and effort is required to develop and maintain the class. Moreover, a class with a large number of methods is likely to be application specific, limiting the possibility of reuse.
- LCOM metric refers to the degree of similarity in methods. Smaller values of mean and standard deviation for MMF indicate design cohesiveness. Moreover, a high value of LCOM indicates that underlying class that might be considered for splitting into two or more classes. The larger the number of similar methods implies a more cohesive class. Cohesiveness of methods in a class promotes encapsulation. It should be noticed that, a low value (near zero) indicates high cohesion. However, a LCOM of zero is not strong evidence that a class has cohesiveness [39].

  For MMF, the values of mean and standard deviation of both WMC and LCOM indicate that its response is fairly uniform with respect to the classical design approach.
- The traditional metrics, LOC, NOC, and NOM which are used as indicator for size of the source code. The larger values of MMF indicate that there is some execution overhead for creation of objects. Also, the A_state pattern distributes behavior for different states across several state subclasses. This increases the number of classes and decreases the degree of coupling. This in turn, affects LOC and NOM. The traditional approach can have smaller values than MMF; however, the proposed MMF is convenient for future maintenance, and later updating.

6. Conclusion

The multimedia mobile framework, MMF, has been designed and implemented by making use of software design patterns. The underlying patterns are integrated and arranged in three levels as follows:

i. First level: contains GoF patterns after adjusting them to Android environment.
ii. Second level: contains visualization patterns after extending them to match MMF.
iii. Third level: introduces two new sound patterns.

The semantic network that emphasizes the inter-relationships between such patterns is given and explained. In order to illustrate the applicability of MMF, it has been used to develop an instance multimedia editor that can be employed to support M-Learning multimedia application.

A quantitative comparison of MMF approach and the traditional object oriented approach has been performed. Such comparison indicates that, the quality of MMF outperformed that of the traditional approach from the view point of encapsulation, cohesion, reusability, flexibility and maintainability.

References:

[1] Ahlawat, S.; Dahiya, R. "A novel Approach Of Text To Speech Conversion under Android Environment". International Journal of Computer Science & Management Studies (IJCSMS), Vol. 13, Issue 5, pp. 189 – 193, July 2013.

[2] Alexander, C.; Ishikawa, S.; Silverstein, M.; Jacobson, M.; Fiksdahl-King, I.; Angel, S. "A Pattern Language: Towns, Buildings, Construction". Oxford University Press. 1977.

[3] Ali, Z.; Bolinger, J.; Herold, M.; Lynch, T.; Ramanathan, J.; Ramnath, R. "Teaching Object-Oriented Software Design within the Context of Software Frameworks". IEEE Frontiers in Education Conference (FIE), pp. 1 – 5, 2011.

[4] Android Developer Site - http:/developer.android.com. Last checked on Jan. 2013.

[5] Bansiya, J.; Davis, C.G. "A Hierarchical Model for Object-Oriented Design Quality Assessment". IEEE Transaction On Software Engineering, Vol. 28, No. 1, pp.4 - 17, Jan. 2002.

[6] Bhati, S.; Sharma, S.; Singh, K. "Review On Google Android a Mobile Platform". IOSR Journal of Computer Engineering (IOSR-JCE) Vol. 10, Issue 5, PP. 21-25. www.iosrjournals.org. Mar. - Apr. 2013.

[7] Bian Wu ; Wang, A.I. ; Hartvoll Ruud, A. ; Wan Zhen Zhang " Extending Google Android's Application as an Educational Tool ". 2010 Third IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning (DIGITEL), pp. 23 – 30, 2010.

[8] Bieman, J. M.; Straw, G.; Wang, H.; Munger, P. W.; Alexander, R.T. "Design patterns and change proneness: An Examination of five evolving systems". Proceedings. Ninth International on Software Metrics Symposium, pp. 40 – 49, 2003 IEEE.

[9] Chun, E.S.; Hong, J. I.; Lin, J.; Prabaker , M.K.; Landay, J. A. ; Liu, A.L." Development and Evaluation of Emerging Design Patterns for Ubiquitous Computing". Conference DIS'04, Boston, MA, USA. ACM Aug. 2004.

[10] Cooper, James W. "The Design Patterns Java Companion". Addison-Wesley Design Patterns Series, Oct.1998.

[11] Desnos, A. "Android: Static Analysis Using Similarity Distance". 2012 45th Hawaii International Conference on System Sciences (HICSS), pp.5394 – 5403, 2012 IEEE.

[12] Ellis, B.; Stylos, J.; Myers, Brad "The Factory Pattern in API Design: A Usability Evaluation". 29th International Conference on Software Engineering (ICSE'07), pp. 302 – 312, 2007 IEEE.

[13] Gargenta, M. "Learning Android: Building Applications for the Android Market". Sebastopol, CA: O'Reilly Media, Inc. Mar. 2011.

[14] Gamma, E.; Helm, R.; Jonhson, R.; Vlissides, J. "Design Patterns, Elements of Reusable Object Oriented Software". Addison-Wesley, 1994.
[15] Gandhewar, N.; Sheikh, R. "Google Android: An Emerging Software Platform For Mobile Devices". International Journal on Computer Science and Engineering (IJCSE), pp 12 – 17, Jan. 2011.
[16] Grace Ramamoorthy "Relevance Of Design Patterns Today, Term Paper". 2011.
[17] Hanafi, H.F.; Samsudin, K. "Mobile Learning Environment System (MLES): The Case of Android-based Learning Application on Undergraduates' Learning". International Journal of Advanced Computer Science and Applications (IJACSA), Vol. 3, No. 3, pp.63 – 66, 2012.
[18] Heer, J.; Agrawala, M. "Software Design Patterns for Information Visualization". IEEE Transactions On Visualization And Computer Graphics, Vol. 12, No. 5, pp.853 – 860, Sep.- Oct. 2006.
[19] Holla, S.; Katti, M. M. "Android Based Mobile Application Development and its Security". International Journal of Computer Trends and Technology, Vol. 3, Issue 3, pp.486 - 490, 2012.
[20] Hudson, S. E.; J. T. Stasko "Animation Support in a User Interface Toolkit: Flexible, Robust, and Reusable Abstractions". ACM Symposium on User Interface and Software Technologies (UIST), pp. 1 – 10, 1992.
[21] Johari, K.; Kaur, A. "Validation of Object Oriented Metrics Using Open Source Software System: An Empirical Study". ACM SIGSOFT Software Engineering Notes, Vol. 37, No. 1, pp.1- 4, Jan. 2012.
[22] Khaer, M.A.; Hashem, M.M.A.; Masud, M.R. "On Use of Design Patterns in Empirical Assessment of Software Design Quality". International Conference on Computer and Communication Engineering (ICCCE), pp. 133 – 137, May 2008 IEEE.
[23] Krasner, G. E.; S. T. Pope "A Cookbook for Using the Model-View-Controller User Interface Paradigm in Smalltalk-80". Journal of Object-Oriented Programming, Vol. 1, No. 3, pp. 26 - 49, Aug. 1988.
[24] Larman, C. "Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development". Third Edition, Addison Wesley Professional, Oct. 2004.
[25] Lee, S. H.; Jin, S. H. "Warning System for Detecting Malicious Applications on Android System". International Journal of Computer and Communication Engineering, Vol. 2, No. 3, pp.324 – 327, May 2013.
[26] Lincke, R.; Lundberg, J.; Löwe, W. "Comparing Software Metrics Tools". ISSTA'08, Seattle, Washington, USA. July 2008 ACM.
[27] Mark L. Murphy. "Beginning Android 2: begin the journey toward your own successful Android 2 applications". Apress 2010.
[28] Mazhelis, O.; Markkula, J.; Jakobsson, M. "Specifying Patterns for Mobile Application Domain Using General Architectural Components". Springer-Verlag Berlin Heidelberg, pp. 157 – 172, 2005.
[29] Mithe, R.; Indalkar, S.; Divekar, N. "Optical Character Recognition". International Journal of Recent Technology and Engineering (IJRTE), Vol.2, Issue 1, pp.72 -75, March 2013.
[30] Padriya, N.; Mistry, N. "Review of Behavior Malware Analysis for Android". ISO 9001:2008 Certified, International Journal of Engineering and Innovative Technology (IJEIT) Vol. 2, Issue 7, pp. 230 – 232, Jan. 2013.

[31] Plakalović D. Simić D. "Applying MVC and PAC patterns in mobile applications". Journal Of Computing, Vol.2, Issue 1, pp. 65 – 72, Jan 2010. https://site.google.com/site/journalofcomputing/

[32] R. Meier, "Professional Android 2 Application Development". Wiley Publishing, Inc., Indianapolis, Indiana, 2010.

[33] Ralph E. Johnson. "Documenting Frameworks Using Patterns". In Proceedings of the OOPSLA '92 Conference on Object-oriented Programming Systems, Languages and Applications, Vol. 27, No. 10, pp. 63-76, ACM 1992.

[34] Reddy, B.R.; Mahender, E. "Speech to Text Conversion using Android Platform". International Journal of Engineering Research and Applications (IJERA), www.ijera.com Vol. 3, Issue 1, pp. 253 - 258, Jan. –Feb. 2013.

[35] Reddy, P. N.; Gyani, J.; Murti, P.R.K. "Design Patterns: A Resource for Reverse Engineering". International Journal on Computer Science and Engineering. (IJCSE) Vol. 02, No. 03, pp. 826 - 830, 2010.

[36] S. Chidamber and C. Kemerer "A metrics suite for object oriented design". IEEE Transactions on Software Engineering, Vol. 20, No. 6, pp.476 - 493, 1994.

[37] Schmidt, D. C. "Experience Using Design Patterns to Develop Re-usable Object-Oriented Communication Software". Communications of the ACM, Vol. 38, No.10, pp. 1 – 10, Oct. 1995

[38] Shanmugapriya, M.; Tamilarasia, A. "Designing an M-Learning Application For A Ubiquitous Learning Environment In The Android Based Mobile Devices Using Web Services". Indian Journal of Computer Science and Engineering (IJCSE), Vol. 2, No. 1, pp. 22 - 30, 2011.

[39] Sharma, A. K.; Kalia, A.; Singh, H. "Metrics Identification for Measuring Object Oriented Software Quality". International Journal of Soft Computing and Engineering (IJSCE), Vol. 2, Issue 5, pp. 255 - 258, Nov. 2012.

[40] Tomáš Chlouba "Design Patterns in Mobile Architectures". 2010.

[41] Vijay K Kerji "Abstract Factory And Singleton Design Pattern To Create Decorator Pattern Objects In Web Application". International Journal of Advanced Information Technology (IJAIT), Vol. 1, No.5, pp. 1 – 9, Oct. 2011.

[42] Zhu, F.; Diao, H.; Huang, W. "Developing Mobile Input Method System Using An Improved Design Pattern Approach". Second International Symposium on Networking and Network Security (ISNNS), pp. 157-160, April. 2010.