Chapter 7

APPLYING ARMMS – AN ARCHITECTURAL EXPERIENCE

7.1 Introduction

Model helps the architect to design the architectures when it is combined with architectural styles. From the architectures, the model evolves by the feedback from the stakeholders out of their architectural experiences. So, the model evolves and in turn the architecture evolves. This is an iterative process.

Reference model for multilingual software also evolved over a period. Different architectures of multilingual software are designed by applying ARMMS combining with different architectural styles like layered, client-server, interpreter, object-oriented, etc. Architectural experience in building the multilingual architectures helped in fine tuning the reference model.

Initially, basic framework for multilingual software, which handles multilingual user interface, multilingual input-output, multilingual file and directory operations, is designed. Multilingual software like Kural, a multilingual programming language compiler and execution unit, Mayan, a multilingual web page design and dynamic construction tool, PONN Anjal, a multilingual email application, PONN SMS, a multilingual SMS utility, etc., are designed by applying ARMMS and using the basic multilingual framework. In this process, the basic framework evolved by capturing more and more multilingual requirements which mainly addresses language aspects.

In this chapter, the various multilingual software, which are designed by applying ARMMS, are briefly explained. An architectural approach to design the multilingual component is portrayed in this chapter. Use-case analysis and domain analysis of selected multilingual software (as examples) are detailed. This analysis presents the software components which addresses the domain or multilingual
functionalities. The architectural experience with these multilingual software developments is presented. Similarly, the experience related to multilingualism is also explained.

7.2 An Overview of the Multilingual Software Development Using ARMMS

ARMMS has been used in designing a variety of multilingual software. This process is an iterative process which helped in fine tuning the ARMMS. The attempts made using ARMMS in developing multilingual software are presented below.

A basic multilingual framework, named 'PONN', has been designed which addresses the multilingual services by the design of multilingual, language and language aspect components. The language aspects like character set, coding scheme and keyboard layout for the Tamil language, which are followed in the development of multilingual software is given as an example in the Appendix B. These components exhibit the functionalities in input-output, user interface and storage aspects. PONN has been developed as multilingual software for providing computer and its services in many languages especially in English and Tamil. The basic multilingual framework has been improved based on the requirements of PONN and a multilingual standard application framework has been designed and developed. This multilingual application framework has been used to develop the multilingual applications like SPECS, KURAL, PONN ANJAL and PONN Chat. Initial development of PONN was done using VC++. Further, PONN has been ported in Java and this version has been tested with different operating systems like Linux and Solaris. Parallel development of PONN has been carried out with more languages like Hindi and Malayalam. PONN has been also extended to work in intra and inter networks as N-PONN and I-PONN respectively.

Design of basic framework has been reused to design and develop the multilingual component based/ web application framework. Development of this framework has been carried out using Visual Basic and Component Programming in Visual Basic with ease due to the design reuse. This framework has been used to
develop applications like MAYAN, a multilingual web page design and construction tool, multilingual VCD lending application and multilingual VISDATA (Multilingual Visual Database Manager).

In a similar fashion, multilingual mobile application framework has been developed and it is used for the application like, PONN SMS and Mobile Banking considering the limited resources like keypad, display etc. These development efforts are hierarchically classified and shown in Figure 7.1. A brief outline about the multilingual software is presented below.

![Figure 7.1: Development Hierarchy of Multilingual Software](image-url)
**PONN**

PONN, multilingual software is developed to provide a working environment for people to work in the language of their choice. This software brought computers a step closer to the common men. Further it extended the computer system to multiple languages. The following features have been provided in the PONN environment over the underlying operating system.

- GUI environment to work with
- Edit multilingual documents and non-documents
- Do File and Directory Operations in different languages
- PONN Multilingual File and Directory Explorer and
- Access Multilingual help on all the above functionalities.

**N-PONN**

N-PONN has been aimed at redesigning the PONN system in Java, a pure Object Oriented Programming language, to make PONN system to run in machines under any platform. The proposed functionality of N-PONN system in a network is attained by adding few sub systems to the existing system which takes care of communication in the network. A common storage is maintained at the server machine and the server is connected to various clients in a network. Various PONN applications can run at the client systems requesting server for various file operations to the storage manager. Network-PONN system enables the user to share files and reduce the cost incurred in maintaining separate storage for individual PONN system. The client and the Server of N-PONN system use TCP/IP to communicate in the network.

**I-PONN**

I-PONN has been designed using the design of the PONN system to suit the Internet arena. A common storage is designed to maintain the PONN multilingual files at the server machine and the server is connected to various clients in internet.
Various PONN applications can run at the client systems requesting server for various file operations. HTTP is the protocol used to share the files in the internet between the client and the server of the I-PONN system.

**SPECS**

SPEcialized Computer System (SPECs) provides a working environment for visually challenged people. It provides a specially designed keyboard layout and also it redefines the usual GUI concept of most software in today’s IT market with voice response functionality. SPECs has been an extension to the Ponn system. It supports for all the functionality of the PONN system to the visually challenged people and also helping to bridge their incapability in using a computer system, by providing the following functionalities

- Provides a mechanism to accept Braille through a specialized keyboard. The keyboard layout will be similar to the six-point Braille layout that the visually challenged user is familiar with. The Braille language is of user’s choice.
- Reproduces the input given by the visually challenged people as voice.
- Produces the output of a program execution as voice in the language chosen or hard copied Braille output
- Produces the error messages and status messages in voice form.

**KURAL**

KURAL is designed as the multilingual programming language. It is designed using the basic multilingual framework to work in the PONN platform. The language is designed as a simple procedural language. It has simple syntax so that the user need not spend extra time and effort to learn the language. It provides the essential features like basic I/O statements, user-defined procedures, etc. It also provides a set of built-in mathematical and graphics procedures.

To translate and execute the KURAL program, a translator and an execution unit have been designed. A very important feature of the translator is that the source
program can be in any regional language supported by the PONN environment. The execution unit can load and execute the translated KURAL program. Input-output of the KURAL program, during the execution of the program can be in a language of the user’s choice. Translate-time and run-time errors, if any, can be displayed in the user specified language.

The following functionalities are provided by the system that will help the user to work in the language of his choice:

- GUI based working environment
- A multilingual editor is used to create document and non-document files.
- Multilingual programming language called KURAL
- **KURAL** programs are coded, translated and executed from the KURAL IDE.
- Manage files and directories in the language of user’s choice
- A comprehensive on-line help system to help users in using the system in the language of user’s choice.

**PONN ANJAL**

PONN Anjal – Multilingual mailing software is developed to provide a working environment for people to send and receive mails in the language of their choice. The functionalities of PONN Anjai software are:

- GUI environment to work in
- registration form
- inbox list
- composer
- outbox for viewing sent mails
- user configuration (views)
- system settings for delivering and transferring
• conversion routines during transfer of mails

**PONN CHAT**

PONN Anjal – Multilingual mailing software is developed to provide a working environment for people to chat in the language of their choice. The functionalities of PONN chat software are:

• GUI environment to work in.
• multilingual registration
• multilingual private and public chat rooms
• user configuration (to save the chats)

**MULTILINGUAL VCD LENDING APPLICATION**

The Video CD Lending application is designed to computerize the activities performed manually in a video CD lending library. This software enables the user to use the system in both Tamil and English. The video CD lending application provides facilities for the customer to select Video CD’s according to his choice of actor, actress, category, language, director, music director and preview the film for making a better selection. It has an interface for returning the borrowed CD’s. It generates a bill for returned CDs. Facilities to add new CD details and artist details are provided. It maintains CD, customer and transaction details for generating reports. All of the above functionalities work in the language of user’s choice.

**MULTILINGUAL VISDATA**

Visdata application is designed to provide facility for the user to create database, design tables, fields and indexes in multilingual. Multilingual data can be added, edited, updated and deleted to and from the table.
MAYAN

MAYAN, A Framework for Multilingual Web Page Creation and Dynamic Construction, developed as a multilingual framework using which a website designer can construct web pages with the same page structure and content in different languages. Using this framework, a designer can store all the versions of a web page together, so that they all have to be updated easily when required. It provides an efficient content management, separating the content from structure.

This framework is developed to support a multilingual interface to help the designer to design web pages in different languages. This framework is provided with necessary web-based components, which can be used in designing a web page and also sent to a browser when there is a request. It is designed to provide support to save the web page into a proprietary format or an XML file or HTML file. Interface Manager offers various interface tools for the designer to interact with the IDE. The Interface Manager has to take care of abstracting the background processes and hiding their complexity from the users.

PONN SMS

PONN SMS is developed to provide the Short Message Service (SMS) in mobile phones in multilingual form. PONN SMS provides a multilingual environment, which facilitates the user to communicate with mobile phone/computer in multilingual form. This multilingual interface has to change according to the user's choice of language.

The PONN SMS system consists of Mobile Client, Web Client, Admin Client, Communication and Server subsystems. The subsystems Mobile Client and Web Client facilitates the user with facilities like composing, sending and retrieving SMS messages, masking unwanted users, redirecting messages, storing their schedules which can be reminded and viewed on demand.
MOBILE BANKING

Mobile banking is the software that is designed to allow all types of banking transactions to be carried out over mobile devices. All banking operations, which do not need the presence of a customer such as ordering a cheque book, balance enquiry, loan eligibility check etc., are done using this software in the user's preferred language.

Sample screen shots of these multilingual software are presented in Appendix C. The next section explains the design approach in order to apply ARMMS in the development of multilingual software.

7.3 Design Approach to Apply ARMMS

Like any other models, ARMMS has been combined with the appropriate architectural styles to build the architectures. This architectural activity is parallel to the design of domain architecture. This architectural activity helps to focus independently on the design of domain and multilingual functionalities of the software. Application of ARMMS in the software development is a very important step which enables to achieve architectural qualities.

In the design phase of software development, domain modeling is a step in the analysis to model the domain components of the software which meet the functional requirements. These domain components are grouped and formed as an architecture using the appropriate model and the architectural style. During the domain analysis, ARMMS is applied as a parallel activity, on the domain components which have the multilingual functionalities. Separation of domain functionalities and multilingual functionalities from the domain component has to be carried out. Two or more separate components have to be designed in order to address these functionalities separately. As a next step, this multilingual component has to be designed by following the module view of ARMMS which follows the layers of ARMMS. The stepwise application of ARMMS is given below

**Step 1:** Identify the domain components which have multilingual functionalities other than the domain functionalities.
**Step 2:** Design the multilingual component, by separating it from the domain component which is identified by the step 1, using the module view of ARMMS.

**Step 3:** Repeat the steps 1 and 2, until all the pure domain components (without multilingual functionalities) and multilingual components are designed.

This 3-step approach helps the designer to have two parallel lines of design and development one focusing on the domain and the other focusing on the multilingual requirements. This yields the qualities like better reuse and easy to modify the domain components and multilingual components. Also, it reduces the time for the design and development of the multilingual software due to the parallel development activity. Core multilingual components are designed and developed during the development process of multilingual software. These components and their design can be stored in the appropriate repositories which can be reused as code level reuse or design level reuse. The application of ARMMS in the design process of multilingual software is illustrated in the Figure 7 2. Consider there is a module C which has been identified during the domain modeling phase and it has both domain functionalities and multilingual functionalities. The separation of domain and multilingual functionalities has to be carried out by the design of the component D, the domain component and M, the multilingual component. The composition of these D and M forms the module C. The multilingual component M has to be considered for further design. M has ‘m’ unique multilingual functionalities and they are carried out by ‘m’ multilingual components. These multilingual components have to be designed with abstract and concrete forms which will help the designer to detach the interface and the implementation of these multilingual components. Consider the multilingual component M1 with specific functionality as an aggregation of language components which meets the same functionalities of the multilingual component in the languages intended for. These language components also have to be designed for the interface and implementation separation. Each language component is designed with the aggregation of language aspect components which are required to meet the functionalities of the language component. Here the language aspect components are reused by many of the language components which belong to the same language.
Module $C$

Domain Module $D$

Multilingual Module $M$

Abstract Multilingual Component $M_1$...

Concrete Multilingual Component $M_1$

Abstract Language $L_1$

Concrete Language $L_1$

Concrete Language Component $LC_{11}$...

Language Aspect Component $LAC_{11}$...

Abstract Language $L_n$

Concrete Language $L_n$

Concrete Language Component $LC_{n1}$...

Language Aspect Component $LAC_{n1}$...

Figure 7.2: Application of ARMMS in the design of Multilingual Module
Figure 7.3: Design of Save Dialog Component using ARMMS
ARMMS based design approach is explained with an example from the PONN multilingual software. Considering the file-save module in a software which is responsible for saving the content in the user specified file, the content which is intended for the storage and the file name specified by the user are the multilingual aspects. Here, the save dialog component gets the multilingual file name from the user, which is considered for the design. Save dialog component is a composition of multilingual window, multilingual label, multilingual textbox and multilingual button components. Abstract and concrete representations of these components are shown in the Figure 7.3 The concrete multilingual textbox component is an aggregation of character input, character display and line-edit components of a language. The abstract components of these functionalities in Tamil language are shown in the Figure 7.3 The concrete implementation of these Tamil components uses the Tamil aspect components like character set, coding scheme, font and keyboard layout.

7.4 Design of Multilingual Software

The multilingual software, discussed in section 7.2, have been designed using object oriented approach using UML. The important step of identifying the components with multilingual functionalities is explained with the help of use-case diagrams, domain class diagrams and the package/architecture diagrams of PONN framework and three multilingual software, namely KURAL, MAYAN and PONN SMS. These diagrams highlight the components/classes/subsystems which have the multilingual functionalities and designed by applying ARMMS. Initially, the PONN framework has been designed with core multilingual components and the other multilingual software have been designed using these components.

7.4.1 Design of PONN Framework

The design of PONN framework comprises of the design of response handler, help handler, file, directory and storage manager and PONN Desktop. The response handler and help handler are the components of Interface manager.
7.4.1.1 Design of Response Handler and Help Handler Subsystems

*Use-case Diagram for Response Handler*

![Use-case Diagram for Response Handler](image)

**Figure 7.4: Use-case Diagram for Response Handler**
The following are the actors of Response handler subsystem and they are explained below.

- **KURAL**: KURAL is a multilingual programming language, which translates and executes KURAL programs. For this purpose, it has to display the translation and run time errors and has to perform various input and output operations, in the required language.

- **FDSM**: File & Directory and Storage Manager loads, saves and creates files/directories on request. On completion or failure of the report it needs to give the result of the request to the user in his native language. To achieve this, it requires the use-case namely *Display Response*.

- **PONN Shell**: The PONN Shell has to display the response message, on completion of failure of the command issued at the shell prompt. The response should be in the language in which the command was issued. To perform this task it requires the use-case namely *Retrieve Message*.

- **PONN Explorer**: PONN Explorer provides GUI environment for performing file/directory and storage operations. On success or failure of these operations it has to display the appropriate response message. For this purpose the PONN Explorer requires the use-case namely *Display Response*.

- **KURAL IDE**: While editing programs in the KURAL IDE editor, to display the necessary messages to the user, KURAL IDE requires the use-case namely *Display Response*.

- **VASU**: While creating or editing a file using VASU it needs to get certain response from the user, such as whether to save the contents of the file or not while closing it. VASU also needs to display appropriate messages whenever required. For these purpose it requires the use-case namely *Display Response*.
The following are the brief discussion of use-cases of Response Handler subsystem:

- **Display Output**: This provides functionalities for displaying various items such as characters, integers, floats, strings, and graphics.

- **Display Errors**: This performs the task of displaying a list of errors, stored in a linked list, on the response window.

- **Get Input**: This performs the task of getting various types of inputs, such as character, integer, real, etc., from the user.

- **Retrieve Message**: This will retrieve the appropriate message from the file, which was loaded using the subsystem FSDM.

- **Display Response**: Displays the retrieved response message on appropriate window.

The use-case diagram for Response Handler subsystem is shown in Figure 7.4.

*Use-case Diagram for Help Handler Subsystem*

All the actors of Response Handler, except the PONN User, rely on the Help Handler for displaying help. The PONN User uses the Help Handler for accessing help on a particular topic. The use-case diagram for Help Handler subsystem is shown in Figure 7.5.

The use-cases for Help Handler subsystem are listed below along with their description.

- **Start Help**: Start Help initiates the help system of a particular type by listing the variable help topics.

- **Select Help Item**: This retrieves the help contents for a selected help topic from the help file, which was loaded using FDSM.
Figure 7.5: Use-case diagram for Help handler
- **Display Help**: This displays the retrieved help contents along with appropriate title on the help window.

- **Link Maintenance**: This maintains a list of help topics retrieved and displayed for each help system.

- **Execute command**: Executes the operations required such as displaying the previous help contents, displaying the next help contents, etc. To provide this functionality, it makes use of Link Maintenance use-case.

The relationship between various domain classes and their interaction with other modules are depicted in the Figure 7.6 and 7.7. The domain classes of Response and Help Handler are as follows:

- **IOCS**: This class is responsible for providing various types of output functions such as displaying characters, strings, numbers and graphics.

- **Error Node**: Error Node encapsulates the details of an error generated during translation or execution of a KURAL program.

- **Error List**: Responsible for maintaining a list of Error Nodes, until all the errors are displayed.

- **Inputs**: This class is responsible for getting various types of inputs from the user, in different languages.

- **Response Handler**: This class provides interface for various response actions such as displaying outputs, getting inputs, displaying various response and error messages and getting response from the user for the displayed messages.

- **Help Node**: Help Node encapsulates the details of a help topic.

- **Help List**: Help List maintains a list of Help Nodes for a particular type of help.
Figure 7.6: Class diagram for Response Handler
- **Help Window:** It is responsible for displaying help contents, for a selected help topic, along with appropriate titles.

- **Selector:** This is responsible for selecting appropriate file names and captions for various GUI elements.

- **GUI Help Handler:** This is responsible for providing GUI based help system. This is a base class from which classes are inherited for each type of the help system. The classes inherited from GUI Help Handler are PONN Help Handler, VASU Help Handler and KURAL Help Handler.
• **Shell Help Handler**: This provides a help system for PONN Shell. The help system for PONN Shell is text based.

### 7.4.1.2 Design of File, Directory and Storage Manager

*Use-case diagram for File, Directory and Storage Manager*

The following are the actors for the File, Directory and Storage Manager. The interaction between the actors and use-cases is depicted in the use-case diagram of Figure 7.8

- **Underlying OS / Storage Driver**: This is an external actor which interacts with the File / Directory operations and Storage Driver. Before the PONN system is loaded, the storage driver performs certain load time routines.

- **Interface Manager**: The interface Manager displays messages that may be errors, warnings, success notification or result of a file, directory or storage operation.

- **KURAL**: KURAL is the multilingual language, which performs translation, and execution of programs written using its IDE. It needs to load the intermediate code whenever execution of a program is requested.

- **VASU**: VASU is an editor, which stores and retrieves data in two modes viz. Document and Non-Document.

- **Explorer**: Explorer provides a visual representation of the file/directory/storage operations along with launching applications.

The following are the use-cases of the File Directory Storage Manager.

- **Send File / Directory Operations Request**: This passes the request issued by the client to the underlying OS.

- **Save File**: This performs the functions of Saving File. The file may be in different formats.

- **Load File**: This loads files of various formats.
Figure 7.8: Use-case diagram for File, Directory and Storage Manager
• **Perform Load Time Routines**: This performs load time check up for the validity of PONN system. It is performed in four passes

• **Execute Shell Mode File Directory Command**: Performs file / Directory operations in the explorer mode

• **Send Response Code**: This creates a response code based on the response of the File / Directory / Storage Manager and passes it on to the Interface Manager.

• **Load Non-Document**: This loads files of non-document format. These files may be one of the map files or the VASU non-document file or a KURAL program

• **Save Non-Document**: Save files of non-document format. Again may be any of the Map files, KURAL program or VASU non-documents.

• **Save IC**: Saves the intermediate code generated by the KURAL translator

• **Load IC**: Loads the intermediate code generated by the KURAL execution unit

• **Save Document**: Save Files of Document Mode.

• **Load Document**: Load files of Document Mode

*Class diagram for File, Directory and Storage Manager*

It had been foreseen that the later versions of the system will lead to changes in both the components in order to incorporate new features. That is, the components that require to do File, Directory and Storage operations plus those components that actually carryout the operations. Some such features are:

• Networking features

• Concurrent requests

• Addition of new Components (of either kind)

• Changes in the interface outlook

• Changes in the internal file and storage structures etc.
Due to the above-mentioned objectives, the two types of components namely the Client and the Server components have been decoupled. This has been attained by inserting a layer between the two types of components, which caters to the two-way communication between them without either of them knowing the internal details about the other. This in-between layer has been implemented by encapsulating the request itself as an Object.

![Class diagram for File, Directory and Storage Manager](Figure 7.9)

Based on the above analysis the following domain classes are identified and they are:

- Request
- FileDirectoryOperationRequest
• StorageOperationRequest
• FileDirectoryManager
• StorageManager
• Client class (For example, Shell, EU, ICG etc)

The key to such an architecture is an abstract Request class, which declares an abstract interface for execute() and getResult() operations and also encapsulates request code and a generic (void) pointer to return result. This is illustrated in Figure 7.9

7.4.1.3 Design of PONN Desktop

Use-case Diagram for PONN Desktop

The following are the actors of PONN Desktop

• PONN user: He is the primary person who is going to launch the request for any component from the desktop. This actor is responsible for loading / closing any component.

• UIM(User Interface Manager): This actor is a class who is responsible for loading the actual component. It is also having the responsibility of providing context switching between components.

• Resource Files: The actor initiates the loading process of any component by supplying the necessary information from the database.

• PONN Component: The actor is responsible for switching between any components. The various PONN components are explorer, KURAL IDE, VASU and shell.

The following are the use cases of Desktop and they are illustrated in Figure 7.10.
- **Load Desktop**: This use case loads the desktop screen with icons, captions, and menus and with shortcut keys. This initiates the process of working with PONN environment.

- **Change Property**: This use case is used to change the current working environment font of the PONN system.

**Figure 7.10: Use-case diagram for Desktop**
- **Save Status:** This use case takes care of saving the current status of each component with switching between components and also saves the current working environment font.

- **Load Component:** The actual loading of each component from Desktop is carried by this use case.

- **Allow Switching:** This use case allows the user to switch between the components that have been loaded already.

- **Unload PONN:** This takes care of unloading the whole PONN system. If any component is currently running while the request for quit is made, it will take care of closing all windows before closing the present window desktop.

**Domain Class Diagram for Desktop**

The **Desktop** consists of domain classes like

- **Desktop Window:** This class is responsible for maintaining the persistent state of the Desktop. Initially it loads the icons, captions for the icons, start button, main menu and shortcut property menu. This class maintains the parent child relationship between this parent Desktop window and other PONN component child windows.

- **PONNEnv:** This class maintains the current Font of the PONN system. Whenever the user changes the Font property of the PONN system, this class updates its state. It also supplies its state details, whenever it's being asked by other subsystems.

- **Singleton:** This class ensures that only one instance can exist for each PONN component. Whenever the UIM class launches any component, an entry for the component is made in this class. If the user is trying to open the same component more than once, it will not launch the same component more than once.
• **UIM (User Interface Manager):** This class performs the task of loading / closing any PONN component from the Desktop

![Class diagram for Desktop](image)

**Figure 7.11: Class diagram for Desktop**

The relationship between each one of them is clearly portrayed in the diagram of Figure 7.11. The system is also interacting with other system classes namely

- IOCS (Multilingual Editor)
- InitPONN (File and Directory Storage Manager)
- ShellWord (Interface Manager package, Shell subsystem)
- VASU (Multilingual Editor)
- ExplorerWindow (File and Directory Storage Manager)
7.4.2 Design of KURAL

**Use-case Diagram for KURAL**

The analysis of the KURAL is documented in the use-case diagram as shown in Figure 7.12. The diagram shows the association between use-cases and the various actors. The following actors were identified for the system:

- **KURAL IDE**: A KURAL program is developed using the KURAL IDE. The IDE helps in writing the program, translating it and finally executing it.
- **File, Directory and Storage Manager**: The File, Directory and Storage Manager load the system file containing the reserved words. The intermediate code file is also stored and loaded by this actor.
- **Interface Manager**: The Interface Manager can also be used to execute a translated KURAL program (intermediate code) This actor handles the...
input-output requirements of the program. It also handles the errors, if any, while translating a KURAL program or executing an intermediate code file.

The following use-cases were identified for the system:

- **Translate**: This use-case is responsible for the translation of a KURAL program. An error-free program is then converted to its equivalent intermediate code. The KURAL IDE, the File, Directory and Storage Manager and the Interface Manager interact with this use-case. The IDE calls the translator, the File, Directory and Storage Manager stores the intermediate code file and the Interface Manager displays errors, if any.

- **Execute**: This use-case is responsible for the execution of a translated KURAL program. The program can be executed from the PONN Explorer and PONN Shell. The File, Directory and Storage Manager loads the program and the Interface Manager displays the output or errors, if any.

**Domain Class Diagram for KURAL**

From the requirements captured, the key classes in the subsystem were identified. The use-cases along with the key concepts were used to form the domain classes for the subsystem. Following are the domain classes identified for the system.

- **Scanner or Lexical analyser (lex)**
- **Language converter**
- **Symbol table**
- **Parser**
- **Intermediate Code Generator or ICG**
- **Execution Unit**

The classes are documented in a class diagram along with their relationships as shown in Figure 7.13. The following explains the domain classes.
- **Scanner or lexical analyser (lex):** The scanner or lexical analyser (lex) is responsible for creating valid tokens from the source program. Errors along with the line numbers, if any are passed on the Interface Manager that takes care of displaying them.

- **Language Converter:** The language converter makes other units of the KURAL system independent of language used for programming. It maintains a list of all the reserved words in various languages. It essentially converts the reserved words in other languages to their equivalent internal language – the language-independent code to be used by the parser and the ICG.

- **Symbol table:** The symbol table is one of the most important data structures for a translator system. It keeps track of the names used by the program and records essential attributes about each. There are two types of symbol tables – the translate-time symbol table and the run-time symbol table. The translate-time symbol table is further categorised into translate-time constant symbol table, translate-time procedure symbol table and translate-time variable symbol table. They are used by the parser and the ICG. The execution unit uses the run-time symbol table.

- **Parser:** The parser uses the lex to get valid tokens. It also uses the language converter to convert the reserved words to the internal language. The translate-time symbol tables are created during Pass 1. Pass 2 validates the syntax and semantics of the program. Errors, if any, along with the corresponding line numbers are once again passes on to the Interface Manager.

- **Intermediate Code Generator or ICG:** The intermediate code generator or ICG generates equivalent intermediate code for the KURAL source program. It uses the list of tokens validated by the parser and the various translate-time symbol tables to generate the intermediate code. The intermediate code is stored on the secondary storage using the File/Directory and Storage Manager for later execution.
Figure 7.13: Domain class diagram for the KURAL

- **Execution Unit**: The File/Directory and Storage Manager load the intermediate code file for the execution unit. The execution unit uses the run-time symbol tables to accomplish its task. The Interface Manager handles the input-output requirements of the program. Errors, if any, while executing the intermediate code are passed on to the Interface Manager.
Package Diagram of KURAL

The foundation for an extensible system lies in a well-designed architecture. The architectural design, a high level design, shows the dependencies between the various packages. A clear and simple architecture should be the primary goal. Figure 7.14 shows the design of these packages. The KURAL package and the associated packages which work with KURAL system are

- **File/Directory and Storage Manager Package:** All the file/directory operations are performed by this package. This package also provides a facility to store/retrieve the files (document, non-document, program and other files) on/from the secondary storage. Its functionality includes ensuring the consistent state of the PONN system file structure.

- **Interface Manager Package:** The Interface Manager package is responsible for most of the user interface. Other packages cooperate with this package for various requirements. Any errors while translating or executing the KURAL program and the input-output requirements of the program are handled by this package. It provides a desktop, an explorer, a shell and a response handler. It also provides a comprehensive help system for the entire system.

- **KURAL Package:** This package basically includes a *translator* and an *execution unit* for the programming language, KURAL. KURAL is a simple procedural language. A KURAL program may be coded in various regional languages. The language has a simple syntax to help beginners learn programming easily. The language supports essential features like basic I/O statements, control statements, user-defined procedures, etc. The language also provides a set of built-in library procedures.

- **Multilingual editor Package:** This includes the editor VASU and the KURAL IDE. VASU can be used to create both document and non-document files. KURAL IDE is used to write KURAL programs. The programs can be translated and executed from the IDE.
7.4.3 Design of MAYAN

The objective of the system is to provide, a Multilingual Framework, which can change the interface according to the users' choice of language, and can be used
to design web pages in different languages simultaneously retaining the page structure and varying only the content of the web components.

The following functionalities will be provided by the system.

- A collection of Web based components.
- A multilingual IDE, which will interact with the designer, and manage the workspace for web page design
- A Page Generator, which will assemble the web page in the language desired by the user
- A Storage Manager, which will store / load designed web pages into different formats.

The IDE - Process Manager should take care of offering and managing the workspace for the design of web pages. The designer client, who makes use of MAYAN for developing web sites in multiple languages, communicates with the system through the IDE - Interface Manager, triggering various actions to be performed by the IDE - Process Manager. The designer has to make use of a repository of web based components, developed as ActiveX controls, for designing the web pages. The IDE - Process Manager should be responsible for maintaining the page structure and contents separately till the designer saves his design permanently.

The IDE - Process Manager has to make provisions for adding client side scripts to the pages designed. It has to allow assigning security levels for the project and its components. Also, it should be capable of switching between various language versions, retaining the page structure, varying only the content of the components. Frame pages have to be designed through the IDE - Process Manager. It has to provide a composite control, capable of allowing other web based controls to inherit its properties. This would help in easy organization and grouping of web based components, making the component level security provided by the system more meaningful.
The IDE - Process Manager is expected to have provisions for developing Custom Controls, as user-developed, reusable tools, which once developed can be stored and used later in several other web pages.

**Use-case Diagram for Designer Part of MAYAN**

In Figure 7.15 the actor using the system is the Designer client. The Designer client is said to be an active actor because he/she is the one who initiates the system. The Designer Client is a person who is working with MAYAN to construct a multilingual web site. The various use-cases are analyzed below

- **Design a Web page:** This is a sequence of interactions between the system and the designer, with the designer placing the required web based components over the work area and manipulating their properties.

- **Design a New Component:** In this use-case the designer works with the web based components to design a Custom Control, which can be used later as a readymade tool for developing web pages.

- **Modify existing component/web page:** When a web page is already developed, it will be stored in XML or Database for later updating. So, to modify a web page, it has to be loaded first. This use-case uses another use-case called Storage and retrieval to load and save a web page/custom control.

- **Create Multilingual Versions:** After completion of the structural design of a web page, its multilingual versions may be created.

- **Define User Levels:** Each of the components used in a web page have a security level assigned to them. The number of security levels for a particular website can be defined and the usage of the components can be restricted as per necessity.

- **Add Scripts:** In a web page, client-side scripts can be added, for which necessary provisions are made by this use-case.
Figure 7.15: Use-case Diagram for Designer Part of MAYAN

- Preview Designed web pages: During the development phase of a web page, the designer needs to constantly check the look of the web page with the browser. For this the HTML page is generated corresponding to the design in the Framework's work area itself. The Storage and Retrieval use-case is referred for this purpose.
- **Storage and retrieval**: When the design of a web page or component is complete it is to be saved, this use-case helps to save and retrieve the web pages.

- **Help**: The designer can use help messages, for efficient usage of the system while developing web pages.

- **Error Handling**: During the design of a web site or custom controls, errors might be caused, which are detected by the error handling use-case.

**Use-Case Diagram for Page Generator of MAYAN**

The Browser client is the actor who interacts with the system requesting a web page constructed using MAYAN, with his choice of language. Figure 7.16 depicts the browser client's interaction with MAYAN.

![Use-case Diagram for Page generator of MAYAN](image)

*Figure 7.16: Use-case Diagram for Page generator of MAYAN*
- **View Pages**: The Browser client might wish to view a web page for which he issues a page request to the Web Server. The following two use-cases generate the requested page, if it has been designed and saved using MAYAN.

- **Generate web pages from Database**: For the web page requested, if the structural details and content are available in the database, the corresponding HTML file has to be generated and rendered to the browser.

- **Generate web pages from XML**: For the web page requested, if the structural details are available in XML, the corresponding HTML file has to be generated and rendered to the browser.

*Class Interaction Diagram for MAYAN*

From the requirements captured, the key classes in the system were identified. The use-cases along with the key concepts were used to form the domain classes for the system. Following are the domain classes identified for the system:

- MyProject
- UserInterface
- WebPage
- CustomControl
- Explorer
- RequestHandler
- StorageManager
- ErrorHandler
- Scripter
- Security
Figure 7.17(a): Class Interaction Diagram - MAYAN
The web based components comprise a collection of components, which include TextBox, TextArea, PasswordBox, PictureBox, LabelArea, LabelBox, CheckBox, RadioButton, CommandBox, ComboBox and HTMLForm.

The classes are documented in a class interaction diagram along with their relationships in Figure 7.17(a) continued in Figure 7.17(b). The classes are shown in rectangular boxes with their names written within the box. The interaction or relation between the classes is depicted with lines drawn from a class to another depending upon the relation between them.

A Relation in which a class is a part of another class or is contained in another class is shown with a solid diamond to the contained class. Such a relationship is called an aggregation. The WebPage and the CustomControl have Composition aggregation with all the WebControls.

The Figure also shows an association relation i.e. "uses" relation between few classes. All these associations are unidirectional i.e. awareness of classes is known one way. The class being used is not aware of the class using it.

The system maintains all the web pages within a class called MyProject, which has an aggregation with WebPage class. WebPage contains the web based components used to construct the web page and the properties of these components such as color, font, position etc., can be inherited from Property class. The UserInterface class keeps the interface according to the user's choice of language.
Figure 7.17 (b): Class Interaction Diagram - MAYAN
Architecture Diagram of MAYAN

The Figure 7.18 represents the overall architecture of the system with two clients, namely the Designer Client and the Browser Client, who are interacting with it.

The Client is equipped with the IDE of the system, which the designer client uses for developing web sites and his Custom Controls. The designer interacts with the IDE – Interface Manager to communicate with the IDE – Process Manager. The IDE – Process Manager offers and manages the workspace for design. A repository of web based components is used for structuring the multilingual pages.

The Storage Manager saves the projects designed permanently in XML or proprietary format, with the page structure and content clearly separated. Communication Manager manages the communication between the IDE – Process...
Manager and the Storage Manager by a mechanism of marshalling and unmarshalling of data. The browser client communicates with the system requesting web pages designed using MAYAN, in his language of choice. The Page Generator combines the structure and content of the web based components and renders to the Browser.

7.4.4 Design of PONN SMS

Use-case Diagram for Client Subsystem of PONN SMS

The PONN SMS, a multilingual SMS System is developed which is composed of two subsystems namely, the Client Subsystem and the Server Subsystem. The first step in the requirement analysis of the Client subsystem is to define the use-cases in order to capture the functional requirements of the system. Use-case diagrams show the various actors and use-cases of the system. These use-cases help in identifying the functionality of each module and its interaction with the actors.

The following are the actors identified for the Client Subsystem.

- **Mobile User**: A primary actor who is going to use the services provided by the system
- **Server**: This actor is responsible for providing services for each service requested by the user

Analyzing the functional requirements of each actor, the following use-cases have been identified for the client subsystem.

- **User-Interface**: This use-case is responsible for getting input from the user and also displaying results to the user. This use-case also takes care of displaying Error/Help message to the user.
- **Mobile SMS Handler**: This use-case is responsible for all SMS related operations of the client which include composing SMS messages, storing, retrieving and displaying SMS for the user.
• **Communication**: This use-case is responsible for sending and receiving SMS to the server process. It also takes care of assembling and disassembling of SMS to be communicated.

The results of the above analysis have been depicted in the use-case diagram of the Client Subsystem in Figure 7.19.

![Use-case Diagram for Client sub-system](image_url)

**Figure 7.19: Use-case Diagram for Client sub-system**

*Use-case Diagram for Server Subsystem of PONN SMS*

The following are the actors identified for the Server Subsystem.

- **Mobile User**: This actor is responsible for sending the service requested by the user to the server subsystem and also to receive the response from the server subsystem.

- **Database**: This actor acts as the store house of all the necessary information needed for the SMS communication.
Analyzing the functional requirements, the following use-cases have been identified for the Server – Administration Services and System Services subsystem.

- **Request Queue**: This use-case is responsible for receiving service requests/messages which are sent in the form of SMS messages from the client subsystem.

![Use-case Diagram for Server – Administrator Services and System Services sub-system](image)

**Figure 7.20: Use-case Diagram for Server – Administrator Services and System Services sub-system**
- **Dispatch Queue:** This use-case is responsible for sending service responses/messages which are sent in the form of SMS messages to the client subsystem.

- **Communication:** This use-case is responsible for sending SMS to the client subsystem. It also takes care of assembling and disassembling of SMS to be communicated.

- **Preferences:** This use-case is responsible for changing the password and the interests of the user.

- **Billing:** This use-case is responsible for handling service usage accounting to the user.

- **Administration:** This use-case is responsible for maintaining user details.

- **Error Help Handling:** This use-case is responsible for handling and maintaining error and help messages

- **Service Type Handling:** This use-case is responsible for adding or deleting a service and also fixing the tariff for the services.

The results of the above analysis have been depicted in the use-case diagrams of the PONN SMS system and the Server – Administrator Services and System Services subsystem in Figure 7.20.

**Domain Class Diagram for Mobile Client of PONN SMS**

The Client Subsystem consists of domain classes like

- **User Interface:** This class is responsible for handling user interaction with the system. It also performs the task of displaying Error/Help messages. It is composed of other classes namely
  
  - KeypadHandler
  - Display
  - ErrHelpMsgHandler
- **Client**: This class is responsible for setting client ID and authenticating the client and the server. It also processes the message received from the server subsystem. It also uses User Interface and Mobile SMS Handler classes.

- **MobileSMSHandler**: This class is responsible for composing SMS message from the user. It is also responsible for storing, retrieving and displaying SMS for the user.

- **Communication**: This class is responsible for sending and receiving SMS to the server process. It also takes care of assembling and disassembling of SMS to be communicated.

The relationship between each one of them is clearly portrayed in the Class Diagram Figure 7.21.

![Class Diagram Figure 7.21: Domain Class Diagram of Mobile Client](image)

**Figure 7.21: Domain Class Diagram of Mobile Client**

The Server – Administrator Services and System Services subsystem consists of domain classes and they are shown in Figure 7.22.
• **Assemble:** This class is responsible for assembling the responses to be sent to the client.

• **Server:** This class is responsible for invoking all classes in the subsystem.

• **Response:** This class is responsible for sending service responses/messages which are sent in the form of SMS messages to the client subsystem.

• **Queue Handler:** This class is responsible for handling the request and the dispatch queue. This class only adds any data to the queue or fetches the data from the queue.

• **Service Handler:** This class is responsible for handling the services with help of corresponding classes like masking, redirection, etc. It identifies the type of service requested and handles the request to the corresponding service.

• **Bill Handler:** This class is responsible for handling service usage accounting to the user.

• **Admin Handler:** This class is responsible for maintaining user details, handling the error/help messages, adding or deleting the services and fixing the tariff for them.

• **Msg Handler:** This class is responsible for retrieving the error code and message from the table.

• **Request Listener:** This class is responsible for listening to the request from the client, disassembling the request and invoking the queue handler.

• **DB Handler:** This class is responsible for identifying the database that is to be accessed for performing the request like masking, scheduling etc.

• **AdminDB handler:** This class is responsible for executing the queries that is related to administrator services.

• **BillingDB handler:** This class is responsible for executing queries related to billing.
Figure 7.22: Class Diagram of Server Subsystem
The package diagram of the PONN SMS system is depicted in Figure 7.23.
Packages identified are:

- **Client Package**: This package is responsible to send the service requested by the user and also to receive the response from the server subsystem.

- **Communication Package**: This package acts as bridge between the client and the server in order to carry out to and fro communication.

- **Server Package**: This package is responsible for maintenance of users along with his profiles. Also, it reminds the user about the bill details. It makes the user login inactive on user's failure to pay the bill. It also performs administrator operations like creating login for user, discarding user, maintain user profiles and personal information, adding/deleting service, maintenance error/help messages and fixing tariff for the services. It also performs system services like identifying the services, handling the database, the request queue and the response queue. This Server package is also responsible for User Services.

7.5 Architectural Experience

The use-case analysis, domain analysis and the architectural design of PONN Desktop, KURAL, MAYAN and PONN SMS presents the various components involved in those systems and the components which have the multilingual functionalities. These multilingual components have been designed using the ARMMS. Architectural aspects derived during the development of these multilingual software are tabulated in Table 7.1

Initially, the basic framework has been designed and it is evolved as PONN Desktop. The layered architecture is used in PONN Desktop, since it followed the ARMMS. An object oriented design approach is used to design the PONN Desktop. Necessary design patterns were used to improve the design. Singleton is used to restrict the multiple launching of PONN client applications. Flyweight pattern is used to handle the multilingual aspects in an effective manner. PONN Desktop has been developed using VC++ under Windows platform. Addition of new language
functions and modification of language functions are easily carried out. Adding a new language does not affect the multilingual components of PONN Desktop due to the usage of abstraction. So, this increases the reusability of the multilingual components. Initial system has been designed with English and Tamil. Later, PONN Desktop is enriched with Malayalam and Hindi. Development of multilingual components of PONN Desktop has been carried out using bottom-up approach with the proper interfaces. So, this approach improves the integrability quality. PONN Desktop has been ported to Java. This process has been completed easily due to the object oriented design and layered approach of the previous system. PONN Desktop is also ported to Linux and Solaris operating systems.

Table 7.1: Architectural Aspects of PONN Desktop, KURAL, MAYAN and PONN SMS

<table>
<thead>
<tr>
<th>Architectural Style</th>
<th>PONN Desktop</th>
<th>KURAL</th>
<th>MAYAN</th>
<th>PONN SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Design Pattern</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Working Platform</td>
<td>Windows/Linux/Solaris</td>
<td>Windows/Linux/Solaris</td>
<td>Windows</td>
<td>Platform with Java Support</td>
</tr>
<tr>
<td>Development Language</td>
<td>VC++/Java</td>
<td>VC++/Java</td>
<td>VB with Component Programming</td>
<td>Java, J2ME and Java Servlets</td>
</tr>
<tr>
<td>Architectural Qualities achieved</td>
<td>Maintainability, Reusability, Integrability and Portability</td>
<td>Maintainability, Reusability, Integrability and Portability</td>
<td>Maintainability, Reusability and Integrability</td>
<td>Maintainability, Reusability, Integrability and Portability</td>
</tr>
</tbody>
</table>

KURAL has been developed on PONN Desktop framework. Most of the multilingual components in PONN Desktop are reused in KURAL. KURAL interpreter follows the interpreter architecture style. Other modules of KURAL inherit the styles like object oriented and layered from the PONN Desktop. In the interpreter
design facade pattern is used to abstract the interpreter from other modules. Architectural qualities of KURAL are derived from PONN Desktop. KURAL is also ported to Linux and Solaris operating systems.

MAYAN has been designed using component based approach. So, the design of PONN Desktop has been reused to construct the component based multilingual framework. MAYAN has been designed with client server style to separate the design of web pages at the client and the storage is at the server. Within the client module, it followed the event driven style. Within multilingual components, the language is handled using the layered approach Facade pattern is used to abstract the client and server, so that they will not have tight coupling. MAYAN has been developed using Visual Basic and VB component programming. Reusability and modifiability are achieved because of ARMMS and component based approach. Integrability is the natural quality in the component based approach. Since VB is used as the development language, MAYAN is restricted to windows platform.

In order to test the ARMMS in mobile (or limited resource) environment, PONN SMS software has been designed. It follows the client server style to separate the multilingual presentation to mobile and multilingual storage to the server. It also followed the event driven approach and layered approach which are derived from ARMMS. Mobile module is designed for multilingual input and output by applying ARMMS using J2ME. Server module is designed with Java and Java Servlets. Facade design pattern helped PONN SMS to achieve the abstraction between the client and server modules. Architectural qualities are achieved with the help of ARMMS.

The multilingual aspects exhibited by these multilingual software are presented in the Table 7.2. Most of the multilingual characteristics explained in chapter 2 have been met. The qualities of multilingual software, which are listed in chapter 3, have also been achieved in these attempts.
Table 7.2: Multilingual Aspects of PONN Desktop, KURAL, MAYAN and PONN SMS

<table>
<thead>
<tr>
<th></th>
<th>PONN Desktop</th>
<th>KURAL</th>
<th>MAYAN</th>
<th>PONN SMS</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. Languages</td>
<td>4 ( Tamil, English, Malayalam, Hindi)</td>
<td>4 ( Tamil, English, Malayalam, Hindi)</td>
<td>2 ( Tamil, English)</td>
<td>2 ( Tamil, English)</td>
</tr>
<tr>
<td>Supported</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Language</td>
<td>Dynamic (at runtime)</td>
<td>Dynamic (at runtime)</td>
<td>Dynamic (at runtime)</td>
<td>Dynamic (at runtime)</td>
</tr>
<tr>
<td>Switching</td>
<td>Static</td>
<td>Static</td>
<td>Dynamic</td>
<td>Static</td>
</tr>
<tr>
<td>Language Management</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Level of</td>
<td>UI and Data</td>
<td>UI and Data</td>
<td>UI and Data</td>
<td>UI and Data</td>
</tr>
<tr>
<td>Multilingualism</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operations</td>
<td>Input, Output, Storage and Process</td>
<td>Input, Output, Storage and Process</td>
<td>Input, Output, Storage and Process</td>
<td>Input, Output, Storage and Process</td>
</tr>
<tr>
<td>Multilingual</td>
<td>Maintainability, Reusability, Understandability, Adaptable and Language Neutrality</td>
<td>Maintainability, Reusability, Understandability, Adaptable and Language Neutrality</td>
<td>Maintainability, Reusability, Understandability, Adaptable and Language Neutrality</td>
<td>Maintainability, Reusability, Understandability, Adaptable and Language Neutrality</td>
</tr>
<tr>
<td>Qualities Achieved</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

PONN Desktop and KURAL have been developed to support English, Tamil, Hindi and Malayalam. MAYAN and PONN SMS have English and Tamil as the working languages. More languages can easily be added to these software. Language switching is dynamic in these attempts. Any time, the user can switch to language he/she wish to work with. Component based development helped to achieve the dynamic language management in MAYAN. The static language management has been used in PONN Desktop, KURAL and PONN SMS. All these software were met the ‘level of multilingualism’ characteristic since these software follow multilingualism at UI and Data level. Multilingual qualities are observed during the design and development stage from the designer perspective and also from the user perspective. Specifically, the language aspect component, language components and multilingual components are reused heavily. Modification of language aspects, which
is frequent, affects only the appropriate components until the interfaces are preserved. These language aspects, language and multilingual levels give a clear picture to the designer to plan for human resources. Also this helps him to communicate the design of software to stakeholders in the early stages of development. Language adaptability is a simple process, since it does not affect the domain components. Also, the domain modules can be designed independently using the abstract multilingual components which are language neutral.

7.6 Summary

This chapter presents the various multilingual software efforts using ARMMS. The design approach to apply ARMMS is comprehensively presented with examples. Design of PONN Desktop, KURAL, MAYAN and PONN SMS are explained in this chapter. Design experiences are stated from the architectural and multilingual perspectives.