Chapter 6

ARCHITECTURE REFERENCE MODEL FOR MULTILINGUAL SOFTWARE (ARMMS)

6.1 Introduction

The stakeholders' requirements, comprising of both functional and non-functional requirements, have to be captured and realized as multilingual software using models. Models can be classified as abstract models and concrete models. Abstract models focus on the core concepts of the intended software leaving out the implementation details. Concrete models reveal the implementation details along with core concepts. Abstract model is also called as reference model.

Reference model is the representation of entities and their interrelationships of the intended software in abstract form. Abstraction in the reference model aids in achieving concrete models. These concrete models are used to build the architectures. Hence reference model acts as a template in building the software architecture. Reference model also helps in explaining the software, prior to its development, to the stakeholders. It can be used as the vehicle for analysis and reasoning. Unfortunately, there is no reference model for multilingual software. Hence it is proposed to form a reference model for multilingual software by refining the aspect based language library model for multilingual software, which was discussed in the previous chapter.

An Architectural Reference Model for Multilingual Software (ARMMS) is derived by applying unit operations. A brief overview about the unit operations is presented in the next section. Application of unit operations on multilingual software is detailed. The layers of ARMMS and the architectural views of ARMMS are described in this chapter. A discussion on the qualities achieved through the ARMMS is presented, which provides a better understanding about the benefits of ARMMS.
6.2 An Overview about Unit Operations

In order to achieve a reference model, a set of design operations has to be applied. These design operations are called as unit operations which are commonly used in software architectures. These software operations are compression, abstraction, resource sharing, uniform decomposition, and replication (Bass et al., 2002; Messina and Meystel, 2002; Kazman et al., 1993) Unit operations are different from architectural styles and design patterns which are more primitive. Hence, unit operations are more abstract than design patterns, farther from implementation. These unit operations are briefed below based on (Bass et al., 2002).

Separation

Separation segregates a part of a system's functionality. Separation places a distinct part of functionality into a distinct component that has well defined interface. It is the most primitive and most common tool of a software architect. Separation may also be used to ensure that changes to the external environment do not affect a component, and changes to the component do not affect the environment, as long as the interface is unchanged. Thus, the operation of separation aids both modifiability and portability. It has two sub types namely uniform decomposition and replication.

*Uniform Decomposition*

Decomposition is the operation of separating a large system component into two or more smaller ones. Uniform decomposition is a restriction of this operation, limiting the composition mechanisms to a small, uniform set. Having uniform composition mechanism eases integration of components and scaling of the system as a whole. Two decomposition mechanisms are distinguished as unit operations and they are presented below:

1. **Part-whole**: Each of a restricted set of subcomponents represents non overlapping portions of the functionality, and every component in the system can be built only from these subcomponents;
2. **Is-a**: Each of the subcomponents represents a specialization of its parent's functionality
**Replication**

Replication is the operation of duplicating a component within the architecture. This technique is used to enhance reliability and performance. When components are replicated, it requires simultaneous failure of more than one component to make the system as a whole fail. As the level of replication in a system increases, the available work can be spread among more of the system’s components, thus increasing throughput.

**Abstraction**

Abstraction is the operation of creating a virtual machine. A virtual machine is a component whose function is to hide its underlying implementation. Virtual machines are often complex pieces of software to create. But once the virtual machines are created, they can be adopted and reused by other software components. This simplifies the creation and maintenance of software components.

**Compression**

Compression is the operation of removing layers or interfaces that separate system functions, and so it is the opposite of separation. When one compresses software, one takes distinct functions and places them together. Compression serves three main purposes and they are to improve system performance, to circumvent layering when it does not provide needed services, and to speed system development.

**Resource Sharing**

Resource sharing is an operation that encapsulates either data or services and shares them among multiple independent consumers. Shared resources, while they are often costly to build initially, but enhance the integrability, portability, and modifiability of systems, primarily because they reduce the coupling among components.

Applying the above unit operations, the steps to design the reference model for multilingual software is explained in the next section.
6.3 Design of Reference Model Applying Unit Operations

Aspect based language library model is a concrete model due to the concrete specification of its structure. It is applied in the development of multilingual software. This experience revealed the need for an abstract model, i.e., reference model for multilingual software to simplify the multilingual software design process. Unit operations are applied to the aspect based language library model for multilingual software to design the reference model. Refined version of the reference model, which is a result of the application of unit operations, is applied in the development of different multilingual software like PONN, KURAL, MAYAN etc. The architectural experience is described in chapter 7. Again, this architectural experience is used for the reference model refinement for the next design iteration. Hence, application of unit operations on aspect based language library model is carried out in an iterative manner. Architectural Reference Model for Multilingual Software (ARMMS) is the outcome of this iterative design process. It is explained below in a consolidated fashion and illustrated in Figure 6.1.

Unit operations are selected based on the architectural decisions. These decisions are used for preserving the existing components or to create newer components. As a first step, uniform decomposition is used to justify the existing components $D$ (domain components) and $M$ (multilingual components) and their relationships. To address the language neutrality, abstraction is applied to divide $M$ (multilingual components) into two sets of components namely, $M_A$ (abstract multilingual components) and $M_C$ (concrete multilingual components). This decomposition detaches the language interfaces and implementations. So, the domain components are developed based on the abstract multilingual components. Hence the language neutrality can be achieved.
Figure 6.1 Design of Reference Model Applying Unit Operations
Uniform decomposition is used to aggregate the language components \((L)\) into the multilingual components \((M)\). Selection of this unit operation is decided to retain the earlier structure in the aspect based language library model. In order to loose couple the language components from the aggregate multilingual set \(M_c\), abstraction is applied to make two set of components namely \(L_A\) (abstract language components) and \(L_C\) (concrete language components). Finally, uniform decomposition is applied to make language aspect components as the parts of concrete language component set. This step is to preserve the existing structure in the aspect based language library model.

### 6.4 Layers of ARMMS

The layers of the ARMMS are directly derived from the hierarchical levels of ARMMS shown in the Figure 6.1. The layers of ARMMS are depicted in Figure 6.2. Language independent domain layer, language neutral domain layer and language dependant domain layer are constructed using separation. Multi-language neutral domain layer is constructed using the composition of multilingual library layer. An aggregation of language library layer forms the multilingual library layer. An abstraction of language library layer is created to separate the interfaces and implementations of language components. Uniform decomposition operation is used to separate language aspects layer from the language library layer. The details about each layer of ARMMS is presented below.

#### 6.4.1 Language Independent Domain Layer

This layer separates the domain software components from the language issues. It helps the designers to concentrate on the domain functionalities. Components in this layer are highly reusable in similar domain applications with different language constraints. This layer helps to design software without language constraints.
6.4.2 Language Neutral Domain Layer

It comprises of multilingual component library and abstract language component library. This layer detaches language issues from the upper layer and also achieves loose coupling between Language Independent Domain Layer and Language Dependent Domain Layer. It comprises of three sub layers, namely

- Multilanguage Neutral Domain Layer
- Multilingual Library Layer
- Abstract Language Library Layer

Multilanguage Neutral Domain Layer

This layer comprises of domain components with multilingual characteristics in a generic form. So, specific language dependencies of the Multilingual Domain components are avoided. Hence, it enables to design language adaptable software which supports effective language management.
**Multilingual Library Layer**

Multilingual component library makes this layer. The multilingual components provide the mechanism for language co-existence and dynamic language switching. Importantly, these multilingual components are independent of a specific language. This layer helps in achieving language neutrality quality. This feature is achieved with the help of effective wrappable operations in multilingual components along with parameterized interfaces.

**Abstract Language Library Layer**

This is dedicated to aggregate the language library in multilingual library in an abstract fashion. It detaches the language specific implementation from the multilingual library components. It also helps in language maintainability qualities in the multilingual software.

**6.4.3 Language Dependent Domain Layer**

This layer takes care of language implementations. Language components are created by the dynamic composition of language aspects. This layer is the foundation layer for any multilingual software with qualities like reusability, maintainability, understandability, adaptability and language neutrality. It comprises of two sub layers and they are

- Concrete Language Library Layer
- Language Aspect Layer

**Concrete Language Library Layer**

Language specific characteristics are designed in the form of language components. Collection of language components forms language library. This layer takes care of realization of language in the software. Hence the language components are designed by the composition of language aspects.
Language Aspect Layer

Language aspects are separated from language components and built as separate components using is-a operation. This increases reusability and maintainability of language aspects and language components. Language aspects are categorized as linguistic, implementation and domain level language aspects as discussed in the previous chapter. They are

- Linguistic level language aspects
- Implementation level language aspects
- Domain level language aspects

6.5 Views of ARMMS

The search for commonalities and principles lead to see that software architecture has four distinct views: conceptual, module, execution, and code. Although there is not yet a general agreement about which views are the most useful, the reason behind multiple views is same i.e., separating different aspects into separate views helps people manage complexity (Oasis, 2007). In order to represent architectural views of ARMMS, conceptual view and module view are chosen. Code view and execution view will be used when the architecture is derived using ARMMS.

6.5.1 Conceptual View of ARMMS

The conceptual view describes the system in terms of its major design elements and the relationships among them. Some of today’s advanced systems were designed with an explicit conceptual view. This view is usually tied closely to the application domain. Some models of conceptual views use communicating objects as the basic design element. Others are used to show the components, connectors and the assemblies of components.

Software systems are generally conceptualized with three subsystems namely, input, process and output. The conceptual view of ARMMS has six subsystems and they are input, process, output and corresponding three support subsystems. This view
is presented in Figure 6.3. Input subsystem takes care of multilingual input requirements with help of multilingual support subsystem for input. Multilingual components which support output subsystem are grouped under the multilingual support system for output. Process subsystem comprises of the components which are supported by the corresponding multilingual support subsystem. These components are responsible for language computations like Translation, Transliteration, Spell check etc. This conceptual view of ARMMS is formed in generic way to suit any software which is intended to support multilingualism.

![Multilingual Software Diagram]

Figure 6.3: Conceptual View of ARMMS

6.5.2 Module View of ARMMS

The module view describes the decomposition of the system. The decomposition is carried out when there is increase in system's size and number of programmers. Abstraction, encapsulation, and the notion of an interface are well-known concepts which are used for decomposition. The decomposition of the system and the partitioning of modules into layers is the main purpose of the module view.

Module view of ARMMS is shown in Figure 6.4. It shows how to logically modularize the multilingual concerns of the software. Each component of the subsystems of the conceptual view can be modularized. Layers are represented with the help of components and required relationships like aggregation, composition and inheritance.
Language independent domain layer has language independent domain components which do not have any multilingual concerns. Language neutral layer comprises of abstract multilingual domain components, concrete multilingual domain components and abstract language dependent domain components. Abstract multilingual domain component can be bound with the domain components which
offer the language neutrality Concrete multilingual domain component is an aggregation of abstract language components. This aggregation can be made during the design-time or execution-time of the software to bind the ‘n’ languages of the stakeholder’s choice.

Language dependent domain layer is made up of concrete language components and language aspect components. Concrete language components are designed with the aggregation of required language aspect components. Different type of language aspect components are domain level, linguistic level and implementation level.

6.6 Discussion

ARMMS is expected to exhibit the multilingual software qualities namely, maintainability, understandability, reusability, adaptability and language neutrality. Considering the design of ARMMS, aspect based library model is the base model which is refined into ARMMS. Also, the structure of aspect based library model is retained in ARMMS. Maintainability and reusability are achieved with the help of the aspect based language library model which is presented in chapter 5. So, these qualities are inherited in ARMMS.

Furthermore, the effect and interactions of unit operations with respect to maintainability, understandability, reusability, adaptability and language neutrality are shown in Table 6.1 based on (Bass et al., 2002) A ‘+’ in the table indicates a positive relationship between an operation and a quality attribute that is, the use of this operation aids in the achievement of the quality goal. A ‘−’ is used in the table to indicate the opposite effect. But the relationship shown in the table does not have any negative effects. Finally, a blank cell indicates that, depending on the context of use, the operation might have a positive or negative effect. Both qualities and unit operations are abstract and only guide us in designer’s choices among design alternatives.
From the Table 6.1, it can be observed that, the abstraction operation has positive effect in all the expected multilingual software qualities. Abstraction is used in loose coupling between the domain and multilingual components. This helps in separating the design of domain and multilingual components. This simplifies the maintenance of both domain and multilingual components. Also, it helps in better understanding of the design of multilingual software by the stakeholders. Separation of abstract and concrete multilingual domain components encourages reusability. Loose coupling due to abstraction offers the adaptation of new languages. Also it enables to design domain components in a language neutral manner.

The part-whole decomposition aids in maintainability, if and only if, the language aspects that change from component to component have been isolated into a single part. Otherwise, the decomposition actually hinders maintainability, because the changes to be made are nonlocal, they are spread out among the language components. Since ARMMS isolates the language aspect components which results in maintainability. It also improves the understandability of the language aspect components. This aspect isolation aids in achieving reusability. Isolation of aspects provides support to adapt newer language aspects based on the stakeholder's requirement.
The 'is-a' decomposition is used to inherit the abstract domain language component into concrete domain language components. Modification of concrete domain language components can not affect the design until the interfaces are maintained. Understanding the language components is simple due to this decomposition. Adopting new languages is made easy by the abstract definition of the language components. This offers language neutrality at multilingual domain component level, which is the aggregation of language components.

So, this relationship between unit operation and multilingual software qualities shows that ARMMS offer the multilingual qualities due to the application of unit operations.

ARMMS is achieved by the iterative design process and versions of ARMMS are used to develop multilingual software. PONN framework is a basic multilingual framework for multilingual software, which handles multilingual user interface, multilingual input-output, multilingual file and directory operations, is designed. Multilingual software like KURAL, MAYAN, PONN ANJAL, PONN SMS, etc., are designed by applying ARMMS and using the basic multilingual framework. This architectural experience reveals the qualities achieved by the ARMMS, which are briefed below.

ARMMS offer modifiability in multilingual software by separating language aspects and they are modeled as separate components. Language level changes of a language do not affect other language components because they are decomposed into separate libraries. Language level changes and multilingual level modification are handled with the help of appropriate abstract models.

Understandability is achieved in ARMMS by the decomposition hierarchy of components into multilingual components, language components and language aspect components. This provides the necessary abstraction into the design and the stakeholders can obtain the information about the layer, they are concerned about, without thinking about the upper layers and bottom layers.
ARMMS offers the reusability at language aspect, language and multilingual levels. Also, this model encourages the design level reuse, by designing the language neutral domain components which can work with the existing language components. This model also acts as a basis to find out the architectural and design patterns for multilingual software development.

Language aspect components offer the choice of choosing the language aspect alternatives to the users. This selection can be done statically or dynamically depending upon the user's requirement. This offers language adaptability to the multilingual software.

Language neutrality of this model provides the development of domain modules and domain related language modules independently. This helps the designer to focus the domain completely and later combine with the language modules. Also, the domain modules can be extended for more languages with ease. On the other hand, the language modules can be reused for similar domain applications.

Based on the above outcomes, ARMMS is compared with mined multilingual models like wrapper, monolithic, multilingual library and language library and the comparison is presented in the Table 6.2. Multilingual software qualities, which are stated in chapter 3, are taken as criterions to compare the ARMMS with mined multilingual software models. ++, + - and - - are three indicators used for the quality-wise comparison. ++ stands for the presence of the specified quality in the particular model. + - stands for the partial presence i.e., quality is present in some aspects and not present in some other aspects. - - stands for the quality is absent in the particular model. These indicators are qualitative in nature.
Table 6.2 Comparison of ARMMS with Mined Multilingual Software Models

<table>
<thead>
<tr>
<th></th>
<th>Modifiability</th>
<th>Understandability</th>
<th>Reusability</th>
<th>Adaptability</th>
<th>Language Neutrality</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wrapper</td>
<td>- -</td>
<td>++</td>
<td>- -</td>
<td>- -</td>
<td>- -</td>
</tr>
<tr>
<td>Monolithic</td>
<td>- -</td>
<td>- -</td>
<td>- -</td>
<td>- -</td>
<td>- -</td>
</tr>
<tr>
<td>Multilingual Library</td>
<td>+ -</td>
<td>+ -</td>
<td>- -</td>
<td>- -</td>
<td>- -</td>
</tr>
<tr>
<td>Language Library</td>
<td>+ -</td>
<td>+ -</td>
<td>+ -</td>
<td>+ -</td>
<td>- -</td>
</tr>
<tr>
<td>ARMMS</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
<td>++</td>
</tr>
</tbody>
</table>

This comparison clearly reveals that ARMMS is better than other multilingual models in offering the multilingual software qualities. Obviously, the ARMMS is useful to the stakeholders in various ways. Here the benefits of the model are provided from the perspective of the stakeholders, especially to the architect, developer and end user.

Architect plays the key role in the software development process. Multilingual software models can be used as communication vehicle by the architect to the stakeholders in order to present an overview about the project. On the other hand, they aid the architect to carry out the software design with ease. Also architect can get different architecture alternatives on combining the multilingual software models with architecture styles. Architect designs multilingual software using the ARMMS by separating domain functionalities from language functionalities. So, the domain components will be designed independently and these domain components are neutral to the languages. Moreover, multilingual software models give a prior knowledge about the software that help the architect in planning the required human resources like computational linguists, language engineers, etc., at the appropriate stages of software development.
Developers also get benefit from the proposed models as they are the work
horses in the software development process. They have to understand the architect’s
to proceed further with the detailed design and development of software.
Multilingual software model gives clarity to the developers in order to progress with
their work at various stages namely, inception, design, development, testing and
deployment. It further guides them in deriving design patterns and reusable libraries
which ease their work.

Multilingual software is designed ultimately for the end users who demand the
multilingualization. Models help the end user in better understanding about the
software during the software development process. The users’ common demands on
the multilingual software are maintainability which provides the modification of
language components and adaptability which provides the alternatives of language
aspects.

In brief, ARMMS offer non functional qualities like maintainability,
understandability, reusability, adaptability and language neutrality which are the
primary concerns of multilingual software. Hence, research hypothesis II is achieved
by the design of aspect based language library model, design of ARMMS and
ARMMS’s quality offerings.

6.7 Summary

This chapter mainly focuses in proposing Architectural Reference Model for
Multilingual Software (ARMMS). Also, an effort is put to present the details of layers
of ARMMS and the views of ARMMS. Also a discussion is made to present the
analysis based on the software qualities which are demanded by the stakeholders and
the impact of ARMMS from perspectives of the stakeholders.