Chapter 4

FORMAL MODELS OF MULTILINGUAL SOFTWARE

4.1 Introduction

IT and IT related services are inevitable in human life. But these services are limited to English. This scenario restricted the benefits of IT to reach the people who do not know English. In order to address this scenario, IT services should be offered in the native languages of the people. On the other hand, internet and IT developments drive the business from local to global market. To meet the demands of globalization, the software have to work in multiple languages. Multilingual software is the one which exhibits multilingualism in one or more aspects like IO, UI, storage, process etc. (Dougnac, 1995; Acharya, 2007; Schmitt, 2001; M17N, 2007). The development of multilingual software is driven by stakeholders of multilingual software namely software engineers, computational linguists, language engineers and end users. The stakeholders’ requirements have been captured and realized in multilingual software. The multilingual software development efforts were made in software industries (Schmitt, 2001; Indiainteractive, 2007), academic institution (Acharya, 2007) and research and development organization (M17N, 2007).

In the software development process, application domain expertise should be captured in order to document the key domain information in the form of a domain model. Also, the model should capture the different views of the software product from the perspectives of relevant stakeholders. Once the model is formed then it can be used as the vehicle for analysis and reasoning. Unfortunately, the existing multilingual developers have not adopted the domain model while developing multilingual software. This has made the multilingual software development more difficult.

In order to overcome the difficulties in developing multilingual software, formal models have been derived by studying the existing multilingual software and
development techniques. These formal models are presented using the following parameters:

- **Description**- gives a generic overview about the model
- **Structural Representation**- explains the formal representation of the models
- **Merits**- points out the qualities that can be achieved from the models
- **Demerits**- lists the missing qualities of the models

A discussion on the multilingual software qualities of these models is also presented, which provides a basis for the future direction of multilingual software development process.

An overview about different formalization approaches of software is presented in next section. The mined models of multilingual software are described in this chapter. Comparison of the mined models based on the multilingual software quality and the inadequacies of these models are also presented.

### 4.2 Software Formalization Approaches

Like any other domain, multilingual software also has architecture either implicitly or explicitly. According to (Perry and Wolf, 1992, IEEE, 2000; Booch et al., 1999), software architecture is a composition of components, connectors and relevant design principles. In (Perry and Wolf, 1992), software architecture is defined by a configuration of architectural elements – components, connectors and data – constrained in their relationships in order to achieve a desired set of architectural properties. But in (IEEE, 2000), architecture is defined by the recommended practice as the fundamental organization of a system, embodied in its components, their relationships to each other and the environment and the principles governing its design and evolution. Significance of the keyword ‘organization’ is presented in (Booch et al., 1999), where software architecture is defined as an architecture which
composes of the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization.

It is clearly evident that software architecture for multilingual software is important like other application domains. In general the steps for achieving architecture directly are hard. Hence, instead of aiming architecture directly, it is desirable to design model(s) which can be combined with architectural styles to accomplish architecture at the qualified juncture. More discussions about the software models are available in (Bass et al., 2002; Parnas, 2001; Clements and Northrop, 2002). As per (Bass et al., 2002), a reference model is a standard decomposition of a known problem into parts that cooperatively solve the problem. If the foundations of domain are firm and unlikely change radically, the model can be static. Otherwise, models will change and evolve. According to (Parnas, 2001), a model is a simplified or reduced size version of a product. Models have some, but not all, of the properties of the “real product” and are used because they are easier to study than the actual product. The views of stakeholders, addressed in (Clements and Northrop, 2002), should be incumbent on whoever is selected to be the repository of the organization’s domain expertise to document the key domain information in the form of a domain model that can be used as the vehicle for analysis and reasoning. Also, the model should capture the different views of the products or from the perspectives of relevant stakeholders.

But these models have to be represented formally in order to better understand and analyze. According to (Shaw and Garlan, 2000), which highlight the values of architectural formalism, it is generally agreed that formal models and techniques for formal analysis are corner-stones of mature engineering disciplines. Formalisms can be used to provide precise, abstract models and analytical techniques based on these models. They are also useful for simulating the behavior of the product. It is reasonable to expect the formalism to have wide variety of users in the area of
software architecture as well. Indeed, several different software artifacts that might be formalized and enumerated are, architecture of a specific system, an architectural style, theory of software architecture and formal semantics for architectural description languages.

Formalization can be achieved by using Z language (Spivey, 1989) or CHAM model (Inverardi and Wolf, 1995). It can also be achieved with UML (Clements et al., 2004; Fowler and Scott, 2000) even though it has some limitations. The mathematical structures (Cohoon et al., 2006; Henno, 2006) are another way of carrying out formalization. In this work, mathematical structures are used in formalization of the models. Reasons for choosing mathematical structures are: they are simple to represent, universal and implementation/technology independent.

4.3 Formalization of Mined Multilingual Software Models

Existing multilingual software inherently have some model but they are not explicitly revealed. Obviously, an effort has been made in this direction to mine the models of the existing multilingual software. The models of multilingual software mined from the literature (Acharya, 2007, Schmitt, 2001; MI7N, 2007; Indiainteractive, 2007; Bateman et al., 1999, Bringert et al., 2004; Lingua, 2007; Fraber-consulting, 2007) are presented below. The mined models are helpful for the software designers and developers to design the multilingual software with ease, based on their demanding requirements. The mined models are wrapper, monolithic, multilingual library and language library.

4.3.1 Wrapper Model

Description:

Core software has been unilingual. In addition to the core software, a wrapper has to be developed. Wrapper is a thin layer which takes care of the mappings of multilingual I/O to and from unilingual I/O of the core software as shown in Figure 4.1 (Fraber-consulting, 2007). Wrappers are specific to the software to which it is intended. Multilingualism in wrappers is much restricted to user interface resources and/or input/output functionalities. Wrapper model is used as adhoc or primitive
solutions to achieve multilingualism in software. Wrapper uses either translation or transliteration techniques to convert the user's language(s) to a language in which the core software works or vice versa.

**Structural Representation:**

Let $S$ be a software and it is defined as a set of components

$$S = \{D, W\} ; D = \{d_1, d_2, \ldots\} ; W = \{w_1, w_2, \ldots\}$$

Where $D$ is the set of functionalities of the core software and $W$ is the Wrapper component set. In order to convert from user's language to core software's language, $m(w_i)$ the mapping function is used which maps $w_i$ to $d_i$.

$$m(w_i) \rightarrow d_i$$

![Figure 4.1: Wrapper Model](image)

**Merits**

Since the solution is primitive, it makes the development process simple. In the situation where the users are in need of quick development, this approach can be used as adhoc solution.

**Demerits**

Due to the limitations of mapping functions, the level of multilingualism is at user interface level. Whenever newer version of software is released, compatible
wrapper has to be developed Non-functional qualities required for the multilingual software are not considered

4.3.2 Monolithic Model

Description:

Domain functionalities and multilingual functionalities are put together in a monolithic component (Dougnac, 1995) as shown in Figure 4.2 This approach makes it hard to modify either domain or multilingual functionalities. Design axioms of modular or object oriented approaches are not considered So, non-functional qualities of the multilingual software are difficult to achieve This approach makes the software as silos. It will be too complex to design and develop

Structural Representation:

Let $S$ be the multilingual software and it is represented as

$$S = \{C\}$$

Where $C$ is the monolithic component having domain functionality as well as multilingual functionalities.

$S$

\[
\begin{array}{c}
\text{C} \\
\end{array}
\]

Figure 4.2: Monolithic Model

Merits

This model meets both the functional and multilingual requirements of the stakeholders, if it works.
Demerits

Major difficulty with this approach is hard to understand the implementation. So, it leads to lack of software qualities like reusability, modifiability etc ,

4.3.3 Multilingual Library Model

Description:

Multilingual functionalities are clearly separated from domain functionalities (Acharya, 2007) as depicted in Figure 4.3. This leads to focused design and development of multilingual software. It also leads towards achieving non-functional qualities which are expected by the stakeholders. Multilingual libraries are hard to design due to intricacies involved in handling multiple languages in each component.

Structural Representation:

Let software $S$ be redefined as

$$S = \{D, M\}; \quad D = \{d_1, d_2, \ldots\}, \quad M = \{m_1, m_2, \ldots\}$$

Where $D$ is the domain component set and $M$ is the multilingual component set. Considering the language aspects, $M$ is monolithic.

![Figure 4.3: Multilingual Library Model](image-url)
**Merits**

Domain functionalities and multilingual functionalities are separated. This leads to the qualities like reusability, modifiability etc.,

**Demerits**

Still, multilingual components are monolithic while considering the language functionalities. This reduces the qualities of multilingual software.

4.3.4 Language Library Model

**Description:**

Individual language library contributes the required multilingual functionalities as shown in Figure 4.4 and 4.5. It aids to carry out language-wise focus on design and development of multilingual software (M17N, 2007) This approach increases the level of non-functional qualities. Proper planning and execution of design and development can be done.

**Structural Representation:**

Let software $S$ be represented as

$$ S = \{D, M\}, \quad D = \{d_1, d_2, \ldots\}, \quad M = \{M_1, M_2, \ldots\} $$

Where $D$ is the domain component set and $M$ is the multilingual component set. Language aspects are separated as

$$ M_1 = \{L_{11}, L_{12}, \ldots\}, \quad M_2 = \{L_{21}, L_{22}, \ldots\}. $$

Where

$L_{11}$ is the required language functionality-1 in language 1,
$L_{12}$ is the required language functionality-1 in language 2, etc.,
$L_{21}$ is the required language functionality-2 in language 1.
$L_{22}$ is the required language functionality-2 in language 2, etc.,
The refined version of language library model is shown in Figure 4.5. Language implementation aspects are separated and modeled as sets. This refinement is represented as

\[ L_{11} = \{L_{111}, L_{112}, \ldots\}, L_{12} = \{L_{121}, L_{122}, \ldots\}, \ldots \]

Where

\( L_{111} \) is the Language functionality-1 in Language 1 using Implementation technique-1 etc.,
**Merits**

Language-wise components can be developed with ease, since the focus during the design and development will be on the particular language. Non-functional qualities are achieved to some extent.

**Demerits**

Language aspects like implementation issues, linguistic issues and domain issues are not considered. This will lead to frequent modification of the components. Also, these issues limit the non-functional qualities.

**4.4 Discussion**

Models are important because they are used to document the key domain information which is captured from the application domain and they capture the different views of the product from the perspectives of relevant stakeholders (Shaw and Garlan, 2000). Multilingual software qualities, which are stated in chapter 2, are taken as criterions to compare the mined multilingual software models and the comparison is presented in Table 4.1. ++, + - and - - are three indicators used for the quality-wise comparison. ++ stands for the specified quality fully present in the particular model. + - stands for the partial presence of quality in some aspects and it not present in some other aspects. - - stands for the absence of quality in the particular model. These indicators are qualitative in nature.

**Table 4.1 Comparison of 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>
</tbody>
</table>
From the Table 4.1 it can be observed that, wrapper model does not perform all the proposed qualities except understandability. Wrappers are generally designed for specific multilingual requirements which should work with the specified software. In most cases, the multilingual aspects are hard coded for the demanded needs. So any modification in the language aspects will affect the wrapper. Wrappers are simple and straightforward to implement the multilingual requirements. Hence, understanding the design and defining the role of the language engineers can be carried out by the designer with ease. As multilingual functionalities are hard coded in the wrapper, the reusability quality can not be achieved. Wrappers are designed with specific language aspects like key board, coding scheme etc. So, they are not adaptable with any other language aspects if the user wishes to add. The wrapper does not support language neutrality quality as it works only with predefined set of languages.

In monolithic model, domain functionality and multilingual functionality are not separated. So, change in domain or language aspects will affect the software. Also, it is really hard to add or modify any functionality due to the non separation of domain and language aspects. Basically, designing the monolithic software is quite complex. So, it lacks in understandability. There are no functions or objects in this model to reuse. All functionalities are packed into single program. So, this model does not support the reusability quality. As this model leads to complex program structure, the addition of new languages is highly complex; hence the adaptability quality is absent. Non separation of domain and language functionalities defeats the quality namely, language neutrality.

Multilingual library model comes with separation of domain and multilingual requirements. Each function or object is designed to handle the given multilingual functionalities. So, this model is also monolithic from the language perspective. Any modification in the language aspects is difficult to carry out due to the monolithic form at the multilingual functional level. Understandability lacks in this model since many language functionalities are put together. Reusability is restricted due to the monolithic nature in the functional level. Adapting new language functionalities is more difficult since each function or object has to be modified. Hard coded languages at functional level restrict the language neutrality.
Language library model handles multilingualism by grouping the individual language functionalities into the multilingual components. Adding new language and modifying the existing language functionalities are simpler. But change in the language aspects restricts the modifiability. Generally, this model is more understandable and the designer can fix up the roles and responsibilities of human resources with language capabilities. Cross cutting language aspects among the language functions or objects limits the understandability. Similarly, reusability is possible at the language level and it is restricted at language aspect level. It lacks aspect level adaptability. Otherwise, it supports adaptability. Language aspects are hard coded. So, this model lacks in language neutrality.

Comprehensively, the mined multilingual models are limited due to the cross cut of language concerns and they lack in the non-functional qualities demanded by the stakeholders. This has made the multilingual software development more difficult. In a nutshell, the following points are the limitations of the mined models.

- Much tightly coupled language aspects like linguistic, implementation and domain related make the system as less reusable and modifiable.
- Each domain with their own language implementation ends up with disjoint systems and reinventing the language implementations by different developers due to different implementation choices.
- This tightly coupled scenario makes the system design and development more complex and limits the understandability.
- Overall, it ends up with an unclear roles of various stakeholder involved in the design and development of multilingual software.

As per hypothesis 1, models of multilingual software are mined and formally represented. Also, a comparison between the models is carried out based on the multilingual software qualities.
4.5 Summary

Like any other domain, multilingual software also needs models which will be helpful in the design process. This chapter mainly focuses in mining different multilingual software models. An effort is put to formalize the models with implementation independent representation. A detailed discussion of these models from the perspective of the stakeholders is presented. These requirements have to be met with a new model of multilingual software and it is presented in the next chapter.