Chapter 2

LITERATURE REVIEW

2.1 Introduction

Information Technology drives many of today’s innovations and offers enormous potential for further innovation in the coming decades. These IT innovations are intended to offer many services to the people. But, they are limited to the people who know English language. In order to provide the IT services to all, the services have to be presented in local languages. The need for localized IT services is very much felt.

Software organizations and research groups have tried to address the software localization activities. But, they are limited to a single language which is intended for. For introducing the language in the computer, the basic requirements like character set, coding scheme, keyboard layouts and rendering techniques have also been addressed by the developers and they are tending towards standardization (Christiansen and Kirby, 2003).

Considering the languages in the world, they can be classified into two sets based on the readiness of the language towards the computer. The first set is a set of American and European languages. These languages are ready for the localization process. This readiness in those languages is achieved due to the similarities in their language structure (Susanne, 2000). The second set of languages like Asian, Arabic and African languages are in the process of getting ready for localization (Knut et al., 2001). These languages have different standards which were created by different ethnic groups. Global standardizations of these languages are in progress. In order to present the issues related to language standardization process, Tamil language has been taken as an example and the standardization efforts made in Tamil language have been presented in Appendix A.
Due to IT innovations, the business has been driven from local to global market. Globalization of business demands that software services should work in more than one language simultaneously. Localization and Internationalization efforts do not address this requirement, since they are making the software to work in a local language, intended for. Hence, multilingual software development has been attempted to meet this requirement.

Research institutions, special interest groups and development organization put their efforts in multilingual software development. Multilingual software efforts have been seen in developing applications like word processor, email software, chatting software, translation software etc. These efforts have been focusing one or more modules like input, output, user interface, storage of multilingual software.

This literature review presents the essential points of multilingual software and its development approaches. Different types of multilingual software are studied from the literature and presented in the next section. Definition of multilingual software and multilingual software characteristics are presented. Classification of multilingual software development approaches are discussed comprehensively. Comparison of these approaches is presented at the end of this chapter.

2.2 Existing Multilingual Software and Its Types

Multilingual printing software and multilingual editor software are the earlier attempts in multilingual software arena (Acharya, 2007; Indiainteractive, 2007; Leap, 2007). Later, many development efforts are made in building translation software and language processing tools like spell checkers, annotating tools etc., with multilingual capabilities (Indiainteractive, 2007; Leap, 2007, Norberto et al, 2005; Felix, 2002). Of late, many software vendors started developing domain applications with multilingual capabilities to meet the demands of globalization. Multilingual software development tools are also available in the literature (Indiainteractive, 2007; Acharya, 2007; GistOra, 2007; Kuppuswami et al, 1999; Prasanna et al, 2002). These efforts are classified into three major categories in order to get a clear picture about the multilingual software as shown in Figure 2.1 and they are
2.2.1 Multilingual Content Creation Software

Electronic content creation can be for an in-house requirement of an organization, or it can be for a web environment. When this content is to be made available to the common man, then it has to be available in the language known to...
him. Also, the traditional and cultural content need to be created to digitize the heritage of the society in their native language and this scenario depicts the need for multilingual content creation (Kuppuswami et al., 2003).

The exchange or transfer of contents between the Government Departments and the people and vice-versa has to happen through the electronic media. Also, the Government has to render the same content in multiple languages in order to reach the entire society. The major impact of the multilingual content creation has been recognized when the traditional Governance shifted to e-Governance (IT-Taskforce, 2007), as the Government has a huge source of information for content generation. In all these cases, multilingualism of the content is an essential criterion.

Multilingual contents are created using multilingual editors. These editors should be capable of handling multilingual input/output and manipulation. Similarly, multilingual web contents are created using web content creation software (IndiaInteractive, 2007, Leap, 2007, Glite, 2007) Usually, these tools are also equipped with multilingual capabilities like spell check, search etc. Machine translation tools (Fraber-Consulting, 2007; SYSTRAN, 2007; Wing Yan Che, 1998) are another approach to create multilingual web contents through which the contents are translated into the required languages and the parallel versions are generated.

2.2.2 Multilingual Processing and Working Environment Software

Software like spell checker, annotating software etc., which are designed to do the language processing in multiple languages are called as multilingual processing software. These multilingual processing software are used based on the multilingual requirements in the software applications and working environments of software. Multilingual editor is the best example which is equipped with multilingual word search and multilingual spell check utilities. Multilingual annotating tools are also used in language based applications. This kind of efforts can be seen in (Leap, 2007; Norberto et al., 2005, Felix, 2002).

Software working environments should support multiple languages to enable the multilingual interface with the computer which is the essential requirement of
multilingual community. This is essential in the countries like India with more than one official language. For example, in Pondicherry, an Union Territory in India, the official languages (Pondicherry Travel, 2007) are Tamil, Telugu, Malayalam and English. Hence, a multilingual working environment is needed for the software in order to provide effective IT services for the people living in Pondicherry, which is an example for multilingual community.

The multilingual working environments are supported by Windows and other commercial software. This working environment provides multilingual interfaces for certain applications like Word processing (ISM, 2007), email clients (Mailjol, 2007; Rediffmail, 2007), mobile clients (ASPSMS, 2007; Dataglyphics, 2007; Multilingual SMS, 2007; Oxygen, 2007, MTNL, 2007; Spice, 2007), etc. But these are only language wrappers provided to the underlying application which can manipulate the language data not as characters but as glyphs. Also, the other interactions of the system, like the messages, help messages are not rendered in the corresponding language. In a system supporting multilingual working environment, the user can choose to interact with it in the language required by him. Accordingly, the input to the system (through the keyboard) and output from the system (messages, processed output, characters typed), user-interface of the system would be configured to the language chosen by the user (Kuppuswami et al, 1999). The multilingual working environment should provide a platform, upon which, multilingual applications can be incorporated and executed.

2.2.3 Multilingual Software Development Tools

Today, most of the programming languages and software development tools are working in the English language only. Software development skills of the non-English communities are blocked by this situation. The multilingual development environment is the next milestone to be reached in the path of multilingual computing. This is because the multilingual development environments help to develop software applications in multiple languages.

Some transliteration efforts in this context can be seen in (PERL, 2007) and (Chen and Hsin-His, 2004), wherein the programmers write programs in languages
other than English. These programs are converted to equivalent programs in English and then compiled. But this does not provide a real multilingual development environment. A multilingual development environment in its simplest form should provide the following utilities to enable software to be developed in the required language (Prasanna et al, 2002).

- A multilingual programming language that can help to write programs in the required language;
- An editor to enter the multilingual program;
- A compiler to compile the program;
- An executor to execute the program

A multilingual working environment forms the basis for this development, as all the input, output, and data representation for the development environment have to be multilingual. Such the multilingual development environments have been attempted in (Schmitt, 2001; Acharya, 2007, Kuppuswami et al, 1999). These multilingual development environments are in their earlier stages and they are limited to support the development of simple multilingual applications.

2.3 Multilingual Software

From the multilingual software endeavors discussed above, it is seen that the multilingual software are being developed by different organizations for their own requirements with their own approaches. Perspectives of the software developers about the multilingual software differ due to the bias towards their specific functional requirements. Comprehensive understanding of these multilingual software development techniques is not possible due to the lack of standard development approaches and formalisms. In order to better understand the multilingual software, the definitions of multilingual software, and their characteristics are derived from the literature and presented below.
2.3.1 Definition

From the literature the following definitions of multilingual software are obtained:

As per J. Dougnac (Dougnac, 1995), multilingual application is typically an application suggesting the end user a list of languages to select based on the user's preference.

According to Schmitt (Schmitt, 2001), multilingual application programs adapt to the language needs of their users smoothly and dynamically, whether the users express the needs directly or indirectly to the application.

Multilingualization in software means that the languages and scripts can be used simultaneously on the software. In order to share information in the whole world, we need a computer environment where support for every language is equally easy (M17N, 2007).

Multilingual software development is based on a set of software tools created to work with text in different languages and scripts (Acharya, 2007)

Consolidating the essential aspects of the multilingual software from the above definitions and adding missing concerns and qualities, we give a new definition to multilingual software is attempted:

"Multilingual software is a software which exhibits domain concerns along with the language concerns in order to enable the users to work in more than one language based on user's dynamic selection of language(s) or configuration of language(s) and the non functional qualities relevant to language."

2.3.2 Characteristics

A detailed study of (Acharya, 2007; AMS, 2007) has lead to the extraction of the following characteristics of multilingual software systems.
(a) **Number of languages supported by the system**

Multilingual software can support more number of languages or less number of languages. The usage of the multilingual software increases when more number of languages supported by it. On the other hand, it also increases the complexity in designing the multilingual software. In the case of less number of languages supported by the multilingual software, the usage of the software decreases and it also decreases the software design complexity.

(b) **Type of switching between languages**

This feature refers to whether users can switch between languages either statically or dynamically. Static switching expects the user to restart the application to switch over to a different language from the current one. Dynamic switching allows the users to switch between languages with ease without the necessity for restarting the application.

(c) **Type of inclusion or exclusion of languages in the system**

Language components and/or languages can be included or excluded from the multilingual software during their compile-time or run-time. Some systems provide dynamic inclusion or exclusion of languages and their necessary modules into an application. Few other systems allow static method of doing the same.

(d) **Fundamental Modules supported by multilingualism**

Software is designed with multiple modules and they are Input/Output module, Process module, Help module, Exception Handling module, Storage module,
etc. These modules are expected to support multilingualism. For example, the error messages that are generated do not make any sense to the user if they are in a language apart from user preferred language. Help modules and Exception Handling modules are also expected to support multilingualism.

(e) Levels of multilingualism

The multilingualism in software can have different levels namely user interface level, data level and combination of both interface and data level. Menu operations, dialogs and input/output windows are expected to exhibit multilingualism at user interface level. Multilingualism at data level means that data can be processed in multiple languages. Multilingualism at user interface level and data level is most preferred level.

2.4 Multilingual Software Development Approaches

Research and commercial initiatives of developing multilingual software are enduring. These initiatives are briefed below in order to gain a better understanding about the multilingual software development. Generally, multilingual functionalities are designed along with the domain functionalities of software. Hence development techniques specific to multilingual software can be seen in the limited form from the literature. In this review, implicit or explicit context of multilingual software development techniques are obtained from the literature and presented here. These development techniques initially used the programming approach. Evolution in software development drives the multilingual software development to use the architectural approach, which is the recent development approach. The classification of multilingual software development approaches is illustrated in Figure 2.2.

The first approach to multilingual software development, similar to traditional software development, is mainly from private vendors, which control the construction of software through each step of product releases. These multilingual software are called proprietary multilingual software and mostly dependent on the vendor who produces it. The capabilities and quality of proprietary multilingual software can be high because of continuous debugging and supporting features consistent with the
expectations of the end-user from the vendor of the software. Proprietary multilingual software and development techniques can be seen in (Schmitt, 2001; Glite, 2007; Indiainteractive, 2007).

Multilingual Software Development Approaches

<table>
<thead>
<tr>
<th>Programming Approach</th>
<th>Architectural Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Structured Programming Approach</td>
<td></td>
</tr>
<tr>
<td>Object Oriented Programming Approach</td>
<td></td>
</tr>
<tr>
<td>Component Programming Approach</td>
<td></td>
</tr>
</tbody>
</table>

Figure 2.2: Classification of Multilingual Software Development Approaches

Like traditional open software, open multilingual software is the other category of multilingual software. It is an approach to software development in which multiple vendors collaborate to build specifications of the technology, independent from proprietary software. The main benefit of open multilingual software is that it provides a uniform terminology of its software structure, which is the foundation for building standard technology appropriate for many end-users. The additional benefit is the interoperability that it may provide between different multilingual software.
applications. Open multilingual software development can be obtained from (M17N, 2007).

2.4.1 Programming Approach

Multilingual software development has followed the popular development approaches of that period. Initially, programming approaches are used for the multilingual software development. Programming approaches are evolved to meet the demands of the hardware and/or software advancements. Popular programming approaches used in the multilingual software development are structured programming approach, object oriented programming approach, and component programming approach. About the various programming approaches used for the multilingual software development are briefed below.

2.4.1.1 Structured Programming Approach

Before structured programming approach, monolithic multilingual applications were developed (Malveau and Mowbray, 2004). This approach has many potential dependencies between the algorithms of the program and does not allow modification easily. If the data representation is modified, there can be substantial impact on the program in multiple places.

Later, with the development of structured programming approach, the multilingual software were developed as small modules which could be reused within the application and also the design of module is simple. In this way, the data aspect of the program and the behavior part were signified separately. This approach allowed development of more flexible and complex multilingual software structures. However, still the dependencies between individual modules and the data are high to build an effective multilingual software system.

Multilingual editor is a very common example for multilingual software using structured approach (Acharya, 2007; Glite, 2007; Prasanna et al, 2002), where editor functionalities are designed along with multilingual functionalities. Mostly multilingual software developers design their own device drivers to input and display.
functionalities. Multilingual translation software, multilingual educational tools are other multilingual software applications using structured approach.

In this approach, common multilingual functionalities are designed as functional libraries and this library is used for different applications. Multilingual input, output and language processing functions are designed as library functions (Acharya, 2007; M17N, 2007; MtStr, 1996). According to multilingualization group (M17N, 2007), a general multilingualization library addresses the multilingual software development. Multilingualization in most of the software is peripheral, that is, multilingual facilities can be isolated from other (main) parts of the software. At the same time, most of the software have common requirement for their multilingual interfaces. A general library that fulfils those requirements makes software development more efficient and inexpensive. It is easy and straightforward to alter existing software into multilingual version using the multilingualization library.

2.4.1.2 Object Oriented Programming Approach

In the next stage, multilingual software development followed the object oriented programming approach like any other software development. This approach supports the development of multilingual software, encapsulating both multilingual data and behavior on it. Multilingual software is designed using abstract data types, called as classes (Booch, 1990). Any changes in multilingual data representation only affect the immediate object that encapsulates the multilingual data. It increases the modifiability quality in the software. So, the multilingual software development, which faces the frequent modification due to the non-standard representation of digital form of the languages, has been adhered in this approach. But, this approach has limitations in managing the objects when a large number of interconnected objects create dependencies.

Object oriented approach for multilingual software development can be seen in (Schmitt, 2001; Chithralekha et al, 2002; Kuppuswami et al, 1999). These design efforts also focus on multilingual functionalities along with domain functionalities. Language and language artifacts are designed as classes. These classes are used for development of multilingual software like Editors, email, Chat etc. The aggregation
of these classes is formed as class library which is used for multilingual software (G-JLET, 2007; Prasanna et al, 2002).

According to Microsoft (Schmitt, 2001), multilingual class library as ML API enables a program to construct text strings representing date, time, money, and other numbers using the rules of a particular locale. Also, this library facilitates comparing, sorting, and converting text using locale-sensitive techniques. In contrast, the ML API is concerned with the external interface – that is the relationship between the application program and the UI devices, especially the keyboard and display. Specifically, the ML API enables a program to easily detect and control the keyboard layout and to find fonts that are compatible with the languages being used within the application.

2.4.1.3 Component Programming Approach

Recent multilingual software development uses the component programming approach which offers more benefits than the object oriented programming approach. Components overlap the properties of object orientation, such as encapsulation and polymorphism, except it reduces the property of inheritance. From the component programming perspective, inheritance is tightly coupled and unsuitable for most forms of packaging and reuse. Instead, components reuse the functionality by invoking other objects and components, rather than inheriting from them. In component terminology, these invocations are called delegations (Malveau and Mowbray, 2004). A brief overview about component programming and how it is used in multilingual software development is presented below.

An overview about Component Programming

A software component is defined as a unit of composition with contractually specified interfaces and their explicit context dependencies. A software component can be deployed independently and is subject to composition by third parties (Chithralekha et al, 2002). It is a group of objects which has a specified interface, working together to provide an application function. Component may refer to many different software constructs, from single application logic to an entire functional
Components have specifications to describe the component encapsulation, which means its public interfaces to other components. The reuse of component specifications is a form of polymorphism. Component specifications are following some standards that are widely reused throughout a system, an enterprise, or an industry. Components may be integrated to create a larger entity, which could be a new component, a component framework, or an entire system which is called composition. The combined component acquires shared specifications from the constituent components. This is often called plug-and-play integration. Reusable components are good reflection of effective software design. Theoretically, components are platform neutral and practically they are restricted to their platform. But they can be used in any programming languages which supports components within their platform.

**Component Programming Approach for Multilingual Software Development**

Advantages of component programming have attracted the multilingual software developers. Specifically, multilingual user interface is developed using component programming approach (Indiainteractive, 2007; Glite, 2007; MUI, 2007). Multilingual text, multilingual label, multilingual form etc., are some of the common multilingual user interface components. Component programming languages uses these components to build the multilingual software. Multilingual education software, Multilingual business applications etc., are examples of multilingual software (Indiainteractive, 2007; Glite, 2007). This approach differs from the previous ones by separating the multilingual functionalities from the domain functionalities.

According to CDAC (iPlugin, 2007), they are focusing their attention on building the application development framework to enable information systems (database applications) to store and retrieve information accurately in local languages. Their components and libraries can be used with third party fonts. The support for
third party fonts is included in the editable components like text box, combo box etc, as well as in the library used to convert between different encodings and fonts.

2.4.2 Architectural Approach

Multilingual software development followed the contemporary software development approach, namely architectural approach in order to gain benefits of that approach. Software architecture gives a high level abstraction which enables the stakeholders to understand the software prior to its development. Architecture based development approaches (Schmitt, 2001, Indiainteractive, 2007) ensures a systematic development of software. Before discussing the architecture based multilingual software development efforts, a brief overview about the software architecture is presented.

2.4.2.1 An Overview about Software Architecture

Software architecture is a promising discipline with a systematic approach towards software development. Key aspects of software architectures are presented below.

Definitions of Software Architecture

Software architecture can be defined in a simple form as a high level design of a software system. It is the organizational structure of the software. More detailed definitions on software architecture with different perspectives are available in the literature and few of them are presented below.

"Software architecture is defined as the structure of the components of a program/system, their interrelationships, principles and guidelines governing their design and evolution over time." (Garlan et al., 1995)

"Software architecture is a description of the subsystems and components of a software system and the relationships between them. Subsystems and components are typically specified in different views to show the relevant functional and non-functional properties of a software system." (Buschmann et al. 1996)
"The software architecture of a program or computing system is the structure or structures of the system, which comprise software components, the externally visible properties of those components and the relationships among them" (Bass et al. 1998)

"Architecture is defined 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." (IEEE, 2000)

**Abstraction**

Abstraction is the key factor in software architecture and it is realized in different levels of software design. According to Pressman (Pressman, 1997), considering a modular solution to any problem, many levels of abstraction can be obtained. At higher levels of abstraction, a solution is stated in broad terms using the language of the problem environment. At lower levels of abstraction, a more procedural orientation is taken. Problem-oriented terminology is coupled with implementation-oriented terminology in an effort to state a solution. Finally, at the lowest level of abstraction, the solution is stated in a manner that can be directly implemented. Different levels of abstractions are:

- **Procedural Abstraction**: is a named sequence of instructions that has a specific and limited function.

- **Data Abstraction**: is a named collection of data that describes a data object.

- **Control abstraction**: implies a program control mechanism without specifying internal details.

The architectural level of design requires a different form of abstraction to reveal high level structure, so that the distinct roles of each element in the structure are clear. The description of software architecture should prescribe the abstract roles of components and their interactions in a software system (Shaw and Garlan, 2000). In brief, abstraction is the operation of creating a virtual machine and that virtual
machine is a component, whose function is to hide its underlying implementation (Edsger W Dijkstra, 1968)

**Software Structure**

Software structure or architectural structure is an important issue of concern (Edsger W Dijkstra, 1968; Parnas et al. 1985). According to Shaw (Shaw and Garlan, 2000), architectural structures are abstract in relation to details of the actual computations of the elements, those structures provide a natural framework for understanding broader, system level concerns such as global rates of flow, patterns of communication, execution control structure, scalability and intended paths of system evolution. Software systems exhibit many structures and in turn defining the overall structure of a system. They are

- Module
- Conceptual or logical
- Process or Coordination
- Physical
- Uses
- Calls
- Dataflow
- Control flow
- Class

Each structure is an abstraction with respect to different criteria. Each structure can reflect its own choice of architectural style i.e components, interrelationships, rationale principles and guidelines (Bass et al 1998).

**Views**

Software architecture is a fundamental organization of a software system in an abstract fashion, comprising many views, which describe the system from different perspectives. Each view of the system describes the composition of components, connectors and data, and their configuration along with the constraints. Architect
needs a number of views of the software architecture for various uses (Perry and Wolf, 1992). According to Hofmeister (Hofmeister et al, 1995), software architecture can be described using conceptual view, module view, code view and execution view and these views are briefed below.

**Conceptual view**

Conceptual view describes the system in terms of its major design elements and the relationships among them. 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 use assemblies of components and connectors. Contrast to this approach, in a few systems, the conceptual view plays a primary role. The module, execution, and code views are defined indirectly with the help of rules. The conceptual view is used to specify the software architecture of system with some attributes to guide the mappings of the other views. The implementation of the system is generated automatically with the help of code generation tools.

**Module view**

The main purpose of the module view is the decomposition of the system and partitioning of modules into layers. When a system increases in size, techniques like abstraction, encapsulation and interface are applied for handling this complexity due to the size and for partitioning work among programmers.

**Code View**

The code view is the organization of the source code into object code, libraries, and binaries. Further, this view organizes source code into versions, files, and directories which strongly affects the reusability of the code and the build time. In the past, source code for a program resided in a single file. In the present scenario, source code is usually split into many files which will have many file types. Object code and binary code are generally mapped into the libraries and/or files.
Execution View

Execution view describes the allocation of functional components to the physical components. If the system has to be distributed, programmers have to decide how to allocate functional components to runtime entities and how to handle communication, coordination, and synchronization among them. Also they have to decide how to map them to the hardware. Now these issues are generally recognized as being architecture level issues. They are better handled by an architect early in the project, rather than being handled by the programmer as the development progresses.

Reference Model

According to Bass et al (Bass et al. 1998), “A reference model is division of functionality together with data flow between the pieces. It is a standard decomposition of a known problem into parts that cooperatively solve the problem.” Reference model arises from architectural experience on the maturity of domain and obtained during the domain analysis. The examples of reference model are user interface reference model (Pfaff, 1985, UIMS, 1992), compiler (Perry and Wolf, 1992), ISO OSI 7-layer model (McClain, 1991), Avionics (Binns and Vestal, 1993) and Mobile robotics (Hayes, 1990, Hayes et al, 1995). Reference Models include system organizations that prescribe specific configurations of components and interactions for specific application areas (Shaw and Garlan, 2000).

Reference model is a concept used in standard conceptual computing models. It is an abstract representation of the entities and relationships involved in a problem space, and form the conceptual basis for the development of more concrete models of the space, and ultimately implementations, in a computing context. It thereby serves as an abstract template for the development of more specific models in a given domain, and allows for comparison between these models. Instances of reference models include, among others: the Open Systems Interconnection Basic Reference Model, the Open Geospatial Consortium reference models, the Von Neumann architecture as a sequential computing referential model, and the Federal Enterprise Architecture reference models (Oasis, 2007)
A reference model is an abstract framework for understanding significant relationships among the entities of some environment, and for the development of consistent standards or specifications supporting that environment. A reference model is based on a small number of unifying concepts and may be used as a basis for education and explaining standards to a non-specialist. A reference model is not directly tied to any standards, technologies or other concrete implementation details, but it does seek to provide a common semantics that can be used unambiguously across and between different implementations (Oasis, 2007).

A reference model is used by architects as a template for composing architectures. Similar to the auto industry in which the logical divisions in the components of a car are made, the software industry uses reference models to make logical divisions and groupings within architectures. Doing so makes it easier for vendor products to be aligned to meet the requirements of the architecture and allows users to understand where their products fit into their corporate architecture. Its working can be compared to how the tyre manufacturer knows that an auto manufacturer understands implicitly that a "wheel" is a round component that bolts to a "hub" and accepts a "tyre" into its rim. Unlike specific architectures, the reference model does not specify what size the wheel is or what bolt pattern it must use, only that it has those attributes. Individual instance of wheel and rim may vary in size, shape, and composition (Oasis, 2007).

The architects use artifacts similar to a reference model in each of these processes, whether explicit or implicit to determine the components that will be part of the solution. It is important to note that architects need not follow the reference model precisely. It is merely a template to start the process. If an industry is well aligned around a common reference model, it benefits all within the industry. Suppliers of specific components can easily describe what their companies deliver in terms of the reference model. Vendors are able to clearly explain what layer they offered from the model. For example, network engineers are able to determine the responsibilities of each layer and avoid architecting networks with functionality duplicated amongst multiple components.
Reference Architecture

Reference architecture is a reference model mapped onto software components and defines the data flows between the components. A reference model divides the functionality, whereas reference architecture is the mapping of that functionality onto system decomposition. The mapping is not necessarily one to one. A software component may implement part of a function or several functions (Shaw and Garlan, 2000). According to Hayes et al (Hayes et al, 1995), “there has been considerable interest in developing reference architectures for specific domains. These architectures provide an organizational structure tailored to a family of applications such as avionics, command and control or vehicle management systems”. Architectural assets, which are yielded from the reference architecture, are shared and reused across organization (M Shaw and D Garlan, 2000, D E. Perry and A. L. Wolf, 1992). Domain Specific Software Architecture (DSSA) comprises

a) A reference architecture, which describes a general computational framework for significant domain of applications.

b) A component library, which contains reusable chunks of domain expertise.

c) An application configuration method for selecting and configuring components within the architecture to meet particular application requirements.

Framework

A framework is a reusable design for all or a part of a software system. Framework is made up of a set of pre-fabricated software building blocks that programmers can use, extend, or customize for specific computing solutions (IBM, 1999). According to Booch (Booch, 1994), “Framework is a collection of classes that provide a set of services for a particular domain; a framework thus exports a number of individual classes and mechanisms which clients can use or adapt. They represent reuse in large”. Essentially a framework is a reusable mini architecture that provides structure and behavior common to all applications of same domain. Frameworks are partially completed software systems that may be targeted at a specific type of
application (Johnson, 1997). So frameworks are generally hybrid of architecture level information and implementation.

**Architectural Qualities**

Architectural qualities are the key concerns of stakeholders of software. These qualities can be achieved effectively during early design phase of the software. Architectures are evaluated by comparing the qualities that they offer. Architectural qualities are listed in (Buschmann et al 1996, Bass et al. 1998) and they are.

- **Reusability**
  Reusability is usually taken to mean designing a system so that the system’s structure or some of its components can be reused again in future for system development. Architectural descriptions support reuse at multiple levels (Garlan et al., 1995; Bass et al. 1998, Gamma et al., 1994; Mettala and Graham, 1992, Allen and Garlan, 1997).

- **Modifiability**
  Modifiability is a function of the locality of any change and it is the ability to make changes quickly and cost effectively. Generally it follows directly from the architecture. Modifications to a system flow from changes in the business needs of the client. They can be broadly categorized as extending or changing capabilities, deleting unwanted capabilities, adapting to new operating environments and restructuring.

- **Portability**
  Portability is the ability of the system to run under different computing environments. These environments can be hardware, software, or a combination of the two. A system is portable to the extent that all of the assumptions about any particular computing environment are confined to one or few component(s).
- **Testability**
  Software testability refers to the ease with which software can be made to demonstrate its faults through testing.

- **Integrability**
  Integrability is the ability to make the separately developed components of the system work correctly together. This in turn depends on the external complexity of the components, their interaction mechanisms and protocols and the degree to which responsibilities have been partitioned. Integrability also depends upon how well and completely the interfaces to the components have been specified.

- **Performance**
  Performance refers to the responsiveness of the system—the time required to respond to events or the number of events processed in some interval of time.

- **Availability**
  Availability measures the proportion of time the system is up and running. It is measured by the length of time between failures as well as by how quickly the system is able to resume operation in the event of failure.

- **Functionality**
  Functionality is the ability of the system to do the work for which it was intended.

- **Usability**
  Usability is defined in terms of learnability, efficiency, memorability, error avoidance, error handling and user satisfaction of software.
- **Security**

Security is a measure of the system's ability to resist unauthorized attempts at usage and denial of service while still providing its services to legitimate users.

### 2.4.2.2 Architectural Approach in Multilingual Software Development

In the existing multilingual software development approaches, domain functionalities are the major concern and multilingual functionalities are the secondary concern. So the multilingual aspects are addressed during the detailed design phase. This limits the abstraction level of multilingual software and in turn limits the qualities of multilingual software. In the evolution of software development approaches, architectural approach got its significance which addresses the software qualities. Architecture-based development techniques have started emerging. Like other domains, multilingual software development also uses the architectural approach. Architecture-based multilingual software development approach are not seen explicitly, even though, the existing software have their architecture implicitly. Architectural approach for multilingual software can be seen in few literatures (Schmitt, 2001; Indiainteractive, 2007).

According to Mithi (Indiainteractive, 2007), they developed a framework in order to deal with the complexity of managing the development of multilingual applications/web sites. This framework extends to cover network and web applications for a range of system architectures and platforms. This framework has been used to create interactive multilingual desktop and networked applications for e-Governance, email, websites, kiosks, office automation and productivity software, etc. It is also be used to develop multilingual applications such as Billing, Public Information Systems, Online forms, Information archives, Message Boards, Chat, and much more.

### 2.5 Comparison of Multilingual Software Development Approaches

Comparison of the multilingual software development approaches are carried out based on the characteristics of the approaches and the qualities achieved using
these approaches. Characteristic criterions are selected based on the design of basic components and how they are interfaced with external world. Criterions selected for the comparison are granularity of the component, coupling, dependencies of the resources, building blocks used in the development and description of the functionality. A set of non-functional qualities are selected as quality criterions which are the requirements of multilingual software. The quality criterions used for the comparison are reusability, modifiability and integrability. Even though the criterions are generic, they have been perceived in the context of multilingualism Comparison of multilingual software development approaches is obtained based on (Yourdon and Constantine, 1975; Booch, 1990, Szyperski, 1998, Bass et al., 1997)).

Table 2.1 Comparison of Multilingual Software Development Approaches

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Structured Programming</th>
<th>Object Oriented Programming</th>
<th>Component Programming</th>
<th>Architecture Based Development</th>
</tr>
</thead>
<tbody>
<tr>
<td>Component Granularity</td>
<td>Very Fine</td>
<td>Fine</td>
<td>Medium</td>
<td>Large</td>
</tr>
<tr>
<td>Coupling</td>
<td>Tight</td>
<td>Tight</td>
<td>Loose</td>
<td>Very Loose</td>
</tr>
<tr>
<td>Dependencies</td>
<td>Compile Time</td>
<td>Compile Time</td>
<td>Compile Time</td>
<td>Execution Time</td>
</tr>
<tr>
<td>Building Blocks</td>
<td>Functions</td>
<td>Classes</td>
<td>Components</td>
<td>Components</td>
</tr>
<tr>
<td>Description of Functionality</td>
<td>Program</td>
<td>Class Declarations</td>
<td>Interface Declarations</td>
<td>Interface Declarations</td>
</tr>
<tr>
<td>Reusability</td>
<td>Very Low</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Modifiability</td>
<td>Very Low</td>
<td>Low</td>
<td>Medium</td>
<td>High</td>
</tr>
<tr>
<td>Integrability</td>
<td>Compile Time</td>
<td>Compile Time</td>
<td>Compile Time</td>
<td>Execution Time</td>
</tr>
</tbody>
</table>
From the Table 2.1, the structured programming approach is more specific to requirements and they are faster to develop. But they lack in the qualities like reusability and modifiability. These qualities are improved incrementally in object oriented programming and component programming. Architectural based approach is the most generic approach among the different approaches. Specifically, the non-functional qualities are achieved in this approach in an effective manner. At the outset, architecture based development offers more benefits to the developers and it is encouraged by the software organizations.

2.6 Summary

The classification of multilingual software is presented. Various definitions of multilingual software are obtained from the literature and they are used for getting a consolidated definition of multilingual software. In order to better understand the demands of stakeholders of multilingual software, a set of characteristics of multilingual software are derived from the literature and presented.

Multilingual software development approaches are the main focus of this chapter. Classification of these approaches has been given to obtain clear understanding about the evolution of multilingual software. Various multilingual software development approaches are presented. On comparison, the architecture based approach for multilingual software development offers more benefits over the other approaches.