Chapter 4

A fresh look at the problem

4.1 Conventional Machine Translation Architecture

A typical machine translation system has a pipeline architecture as shown in figure 4.1.

This architecture has inbuilt ‘fragility’ and hence will never work with 100% accuracy. The main reason is modules like Parts Of Speech (POS) Taggers and Parsers do not give 100% performance and further the errors get cascaded leading to lower level
Chapter 4: A fresh look at the problem

of performance. The best POS taggers for English give correct tags only 95-97% times. This means in one page text with around 300 words, approximately 9 to 15 words get wrong POS tags leading to wrong choice of meaning and thereby wrong translation. The best parsers for English available in the world give the first correct parse only 30-40% of the times. Thus on an average only one sentence out of three sentences is parsed correctly. The other way of measuring the performance of the parsers is in terms of relations. When a parse goes wrong, it is not that machine fails to recognise the sentence completely. It was observed that machine identifies many of the relations correctly and it is only a few relations where machine goes wrong. Hence a dependency based method for measuring the performance was suggested [59]. The best parsers in the world have around 90% performance. Translation further involves word sense disambiguation (WSD). WSD typically involves an analysis of the context, and thus heavily depends on the parsed output and the POS tags. Furthermore, since the language string does not ‘express’ everything explicitly, WSD also has to depend on the extra-linguistic information. This further reduces the accuracy of the translation. Since in this architecture machine makes a choice before producing any output, and the criteria for these choices are ‘heuristically governed’, there are chances of making mistakes by machine which are not ‘human controllable’. Finally a pipeline architecture without any parallel outlets, cascades the errors and the architecture itself does not provide any fall back mechanism.

4.2 Re-visiting Translation

It is amazing that even after more than 50 years of repeated attempts, interest in MT continues to grow. The main reason behind this is the increasing NEED of the society for the translation. With the advent of World Wide Web, information is now available at the click of a button. In this age of Information, knowledge is power. Information access is a human right. As is evident today, most of the information on the web is
available only in English. In India more than 90% of the people do not understand English. This naturally will lead to a digital divide unless this information is made available to the others in their mother tongue. The amount of digital information created is so enormous that it is impossible to think of getting it translated manually.

What people really need, in this era of information, is the possibility of ACCESS to the information in other language. Anusaaraka provides an architecture to build a Machine Translation system, which in addition to providing automatic translation, also provides an ACCESS to the original text at various levels of coding such as word level, word-group level, sentence level, etc. assuring the faithfulness. We explain the concept of an ACCESSOR with an illustration of a Script Accessor followed by the concept of a Language Accessor.

4.3 An illustration of a script Accessor

A script accessor allows one to transliterate a text in one script into the other. For example, the IAST (International Alphabet for Sanskrit Transliteration) (see appendix J) is the most popular transliteration scheme used in the publications of texts related to Sanskrit, Pali, and other Modern Indian languages. This scheme uses various diacritic marks to represent the text faithfully in roman. The Graphics and Intelligence based Script Technology (GIST) is an example of a computational tool for script accessor. GIST technology provides a faithful conversion of any Brāhmi based modern Indian language text into Devanāgarī. In order to remain faithful, Devanāgarī script has been enhanced by introducing some extra graphemes (see appendix K). Figure 4.2 displays two Telugu words in Telugu script. Figure 4.3 shows the same Telugu words in an extended Devanāgarī script, transliterated by the GIST terminal. The advantage of this enhanced script is that, one can now type a Telugu text in Devanāgarī and still would be confident enough to revert back to the Telugu
script at the press of a button.

Figure 4.2: Telugu text in Telugu script

Figure 4.3: Telugu text in the extended Devanāgarī Script

The salient features of this enhanced notation are:

• faithful representation,

• reversibility,

• no loss of information,

• use of additional but minimum number of special characters,

• text in an unknown script is accessible with a little extra training.

Thus the GIST technology helps us to overcome the script barrier. The cost is: one has to put in some extra effort to learn the enhanced Devanāgarī. The effort involved in learning a few graphemes is much less as compared to the effort involved in learning a new script.

4.3.1 Difficulties in developing a script accessor

It is not always easy to develop a script accessor. Telugu and Devanāgarī being the scripts originated from the same Brahmi script it was an easy task to develop a Telugu script accessor by enhancing the Devanāgarī script. Similarly to develop a
transliteration facility from Hindi to Telugu, one needs to enhance the Telugu script.

But in order to develop a transliteration tool from say English into Devanāgarī, it is necessary to couple the script with the pronunciation and then transliterate into extended Devanāgarī. The heteronyms (words with same spelling but different pronunciation) are the problem sources. Such words being a few in number, the task is still doable.

When it comes to Urdu, the situation is a little more complex. Urdu and Hindi together as Hindustani, collectively form the third most populous language in the world. Both have Indian origin and have drawn from Sanskrit through Śauraseni, Apabhraṃśa and Khadi Boli. The syntax of both languages is almost the same and there are many words and expressions commonly used in both the languages. Maisica [63] points out that they are not even two dialects: they are exactly the same dialect used by two different communities. It is well known that Premchand wrote his stories in Urdu script and got them transcribed into Devanāgarī. The common language with common vocabulary is referred to as Hindustani that could be written in both the scripts that is Devanāgarī and Perso-Arabic. Use of two scripts for Hindustani has divided the world of Hindustani into two. Urdu uses Perso-Arabic script while Hindi uses Devanāgarī (see appendix L).

The major reasons why Urdu-Hindi transliteration is not simple is:

- Urdu does not use halanta,
- Typically Urdu does not use diacritic marks to represent short vowels,
- The semivowels vāva, bādi ye and chotī ye are sometimes used as long vowels and sometimes they are used as consonants.
Chapter 4: A fresh look at the problem

So a user reading the ‘faithfully transliterated’ text has to put in extra effort. In case of a semi vowel, the reader has to decide in the context whether it is a consonant or a vowel. In other cases s/he has to supply the missing vowels, and also identify the conjunct clusters and their boundaries. Figure 4.4 illustrates these difficulties. Figure 4.5 shows a glimpse of faithful transliteration leaving the burden of deciphering to the human reader who has the desired lexical knowledge.

![Figure 4.4: Urdu-Hindi transliteration problems](image)

![Figure 4.5: Urdu-Hindi faithful transliteration](image)
4.4 From Script Accessor to Language Accessor: A fresh look at the problem

The task of extending the concept of ‘accessor’ from ‘script accessor’ to ‘language accessor’ is much more complex. Apart from learning a script, the learning of a language involves learning one or more of the following:

- Spelling
- Pronunciation
- Vocabulary
- Morphology
- Syntax
- ...
- New Concepts
- Culture

Just as the script accessor reduces the burden of learning a script, the language accessor is planned to reduce the burden of learning a language. Various computational tools such as morphological analysers, parsers, and disambiguation tools such as POS taggers, WS Disambiguators exist. The question is design of a proper architecture which provides a faithful ACCESS. Let us look at the problem of developing a language accessor afresh, taking stock of available resources and also the capabilities of the computers and humans.

1. Machines are equipped with large memory storage and high speed computing power, whereas humans are good at interpretation. So the natural suggestion is why not share the load between man and machine?
The question is how to share the load between man and machine?

2. Some of the components like morphological analyzers and dictionaries in principle can produce “accurate” information, whereas some other components like POS taggers, parsers in principle are prone to errors.

The question is how to separate the more reliable components from the less reliable ones, and assure graceful degradation.

3. During the course of translation there exists a tension between faithfulness to the original text and naturalness in the target language making accurate translation a difficult task. Machine translation systems have a tendency to favor naturalness over the faithfulness. These systems, therefore, serve only a certain strata of people. But, there are others who would like to have an access to the “original” text (quality of faithfulness) without any “distortion” as introduced by the translation process.

The question is, can the diverse needs of these different strata of people be addressed?

4. Finally, a machine translation system requires many linguistic tools and resources. Most of them are available for English for free download on the Internet under General Public License (GPL). It is natural to make use of these resources rather than reinventing the wheel. In fact there are more than one parser, POS tagger and morphological analyzer for English available under GPL.

How should the system be designed so that one can plug-in different language tools such as POS taggers, parsers, morphological analyzers, etc.?

4.4.1 Earlier efforts

The earlier efforts of building Language Accessors (anusaarakas) among the Indian languages have answered the first and the third questions. The first anusaaraka system
was built in the early 90’s from Kannada to Hindi [70]. The claim of the anusaaraka
was that it is possible to overcome the language barrier in India using anusaaraka.
The Kannada-Hindi anusaaraka demonstrated how one can take advantage of the rel-
ative strengths of the computer and the human reader to build a practical system.
Special notational devices were devised to bridge the gaps between Kannada and
Hindi at the word as well as sentence level. This introduced some amount of burden
on the reader. But this at the same time provided ‘faithful’ image of the original
text into the target language, which any serious reader would like to have. For the
casual readers, anusaaraka provided a post-editing tool, with which a Hindi editor
can edit the text semi-automatically, and generate a text which sounds natural to a
target language reader. Possibility of using custom-made dictionary allowed a user
to produce output of his choice.

Four more anusaarakas from four Indian languages viz. Telugu, Marathi, Punjabi and
Bengali into Hindi were built in the next few years. The alpha versions of these four
systems demonstrated the feasibility of the extension of different pairs of languages.
The output of these anusaarakas ‘closely followed’ the source language constructions
and the source language semantics, thereby influencing the output. The more the
source language is closer to the target language, lesser is the effort a human reader
has to put in understanding the text and the more close is the output to the target
language.

The lack of sufficient corpus, lack of bilingual as well as monolingual linguistic re-
sources necessary to build computational tools and lack of motivation or necessity to
access texts in other Indian languages put a temporary check on the further develop-
ment of these anusaarakas among Indian languages.
4.4.2 Expanding the horizons

The advent of World Wide Web, availability of enormous amount of information in English, availability of several computational Linguistic resources for English under general Public License (GPL) opened up new vistas for the anusaaraka group to develop the English-Hindi anusaaraka. The second and fourth questions raised above thus become very relevant in the context of English-Hindi anusaaraka. With the availability of various computational tools for analysis of English and also libraries for developing GUI easily, the following guidelines emerged naturally influencing largely the architecture of anusaaraka.

4.5 Anusaaraka Guidelines for developing a MT system

- Make complete information available to the user but, do not clutter the scene.
- Separate the resources that can be made, in principle, reliable from those that are, inherently unreliable. Mention explicitly the degree of reliability.
- Provide alternative means to get the information in case it is not already made available.
- Do not reinvent the wheel.
- Use existing resources and tools.

How do we achieve this?

- Make complete information available to the user but, do not clutter the scene.
  — Ensure Substitutivity and Reversibility
  — Hiding Mechanism/User Interface
• Separate the resources that are, in principle, reliable from those that are, inherently unreliable. Mention the degree of reliability explicitly.
  — Run various modules in parallel, and show the output at various levels in descending order of reliability.

• Provide alternative means to get the information
  — Provide online help,
  — Develop Human readable algorithms first and then implement whatever is possible with today’s technology and resources.

• Do not reinvent the wheel
  — Resort to GPL

• Use existing resources and tools
  — Develop suitable interfaces for ‘plug and test’ of different tools,
  — In case several tools are available for a particular task, use voting to select the best output.

4.6 Architecture of the Anusaaraka system

The Anusaaraka system has two major components.

1. anusaaraka engine, and

2. User-cum-developer interface.

Anusaaraka engine is the main engine of anusaaraka. This engine produces an output in different layers making the process of Machine Translation transparent to the user. The architecture of anusaaraka system is shown in Figure 4.6. This architecture differs from the conventional architecture in three major ways—

1. Various language analysing modules for Source language such as morphological analyser, POS tagger, Chunker, Parser, etc. are run in parallel. These modules,
along with the knowledge of the contrastive features between English and Hindi and handcrafted rules for WSD, are used for WSD and determining the phrase boundaries, identifying the phrasal head, etc. The image of source language is shown at various levels of information encoding such as word level, phrase level and sentence level.

2. A graphical user interface has been developed to display the spectrum of outputs. The user has flexibility to adjust the output as per his/her needs.

3. Special ‘interfaces’, which act as ‘glue’, have been developed for different parsers, which allow plugging in of different parsers thereby providing modularity.

### 4.7 Anusaaraka engine

The current anusaaraka engine has five major modules viz.

1. Word Level Substitution

2. Word Grouping
Figure 4.7: The core anusaaraka

3. Word Sense Disambiguation

4. Phrase Boundary marker

5. Target Language word order generator

Each of the above five modules is described below in detail. A justification of how these modifications are appropriate in answering the questions raised in section 3.4 is presented. The concepts involved behind each of these modules are language independent. For the ease of putting across the concepts, we have used English-Hindi language pair for which the system has been tested.

4.7.1 Word Level Substitution

Every word is split into two parts – a base and a suffix. This module provides a ‘gloss’ of each source language word (i.e. the base and its suffix) into the target language. To provide a ‘gloss’ of polysemous words without compromising on the ‘faithfulness’ is a challenge. We discuss the solution anusaaraka offers to handle the polysemy.
4.7.2 Concept of Padasutra

By looking at various usages of any polysemous word, one may observe that these polysemous words have a ‘core meaning’ and other meanings are often natural extensions of this core meaning. In anusaaraka an attempt is made to relate all these meanings and show their relationship by means of a formula. This formula is termed *Padasūtra*. (The concept of Padasūtra is based on the concept of ‘pravṛttī-nimitta’\(^1\) from Indian Grammatical Tradition.) The word padasūtra itself has two meanings:

1. a thread connecting different senses of a pada,
2. a formula specifying the meanings of a pada.

Here is an example of Padasūtra:
The English word *leave* as a noun means *chutti* in Hindi, and as a verb its Hindi meaning is *choḍa*. We see that the two meanings are related. *Leave* basically is a verb and the corresponding noun gets derived from this verb. Hence the Padasūtra for *leave* is

\[
\text{leave: } \text{choḍa}[\rightarrow \text{chutṭi}]
\]

Here ‘\(a[\rightarrow b]\)’ stands\(^2\) for ‘\(b\) is derived from \(a\)’.

The concept of padasūtra takes care of two basic principles of anusaaraka viz. **substitutivity** and **reversibility** [70]. Substitutivity ensures that all the meanings of the source language word have been taken care of. Or in other words, irrespective of the context, one can substitute the padasūtra for a given word. The reversibility ensures that the substitution does not create ambiguity. At this level, machine takes the load

---

\(^1\)Refer to the chapter 8 for more details.

\(^2\)for the notation used in the padasūtras, refer to the appendix B
Chapter 4: A fresh look at the problem

of morphological analysis and the lexical transfer. And the load of interpreting the
text by selecting appropriate choices taking into account the context of the source
language lies with the reader. Thus, by division of workload and adoption of the
concept of ‘padasūtra—word formula’, the approach guarantees that the first level
output is ‘faithful’ to the original and also acts as a ‘safety net’ when later modules
fail. Table 4.1 shows an example of the output after substituting the word formulae.

<table>
<thead>
<tr>
<th>He</th>
<th>kept</th>
<th>the</th>
<th>book</th>
<th>on</th>
<th>the</th>
<th>table.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vaha{pu.}</td>
<td>rakha{ed/en}</td>
<td>the</td>
<td>pustaka</td>
<td>on</td>
<td>the</td>
<td>tēbala.</td>
</tr>
</tbody>
</table>

Table 4.1: padasutra-output

At this level some of the English words like function words, articles, etc. are not
substituted. The reason being they are either highly ambiguous, or there is a lex-
ical/conceptual gap in Hindi corresponding to the English words (e.g. articles), or
substituting them may lead to catastrophe. For example, if prepositions are substi-
tuted at this stage, it may result in a catastrophe as shown in table 4.2.

<table>
<thead>
<tr>
<th>He</th>
<th>kept</th>
<th>the</th>
<th>book</th>
<th>on</th>
<th>the</th>
<th>table.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vaha{pu.}</td>
<td>rakha{0/ed0/en}</td>
<td>pustaka</td>
<td>para</td>
<td>tēbala.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Catastrophe

It is likely, just by looking at the Hindi layer - the proximity of pustaka and para,
that one may interpret it as ‘He kept the table on the book’.

4.7.3 Training Component

To understand the output produced in this manner, a human being needs some train-
ing. The training presents English grammar through the Pāñinian view. Chapter 6
describes the methodology followed to arrive at the contrastive English Grammar
a Hindi speaker has to acquire in order to follow the anusaaraka output faithfully. Thus, if a user is willing to put in some effort, s/he has complete access to the original text. The effort required here is that of making correct choices based on the common sense, world knowledge, etc. This layer ensures that the layer produces an output, which is a ‘rough\(^3\)’ translation that systematically differs from Hindi. Since the output is generated following certain principles, the chances of getting mislead are less. Theoretically, the output at this layer is reversible.

4.7.4 Word Grouping

Sometimes a group of words trigger a new meaning. This is typically the case with compounds and idioms. For example, the group of words ‘the big cat’ refers to ‘animals belonging to the cat family who are able to roar and live in the wild’. While translating into Indian languages naturally one can not get the correct sense if it is translated as ‘bādi billī’. Similarly, the verb groups – sequence of auxiliaries followed by a main verb need to be translated as one unit. For example, the verb group ‘are going’ needs to be translated as one unit, as jā rahā hai\{ba.\}, and not individually separately.

It is possible that in some cases more than one ways of groupings is possible, as illustrated below.

They (are playing) cards.
These are (playing cards).

In such cases, the system should produce both the answers. Of course, in such cases, we know that the words in the proximity provide the clues. But, in principle, the information may not be available in the close proximity. The later modules use the parser and may produce the correct answer. But in case parser fails, then this layer

\(^3\)rough here means rough as in rough journey, where the journey takes you to the destination, though it is uncomfortable
provides possible answers to the users, from which the user can select the correct one. Each of these layers ensures that there is a fallback mechanism in case the later modules fail.

### 4.7.5 Word Sense Disambiguation (WSD)

The next module in Amusaaraka is the Word Sense Disambiguation. The WSD task may be split into two classes:

1. WSD across POS
2. WSD within POS

The POS taggers can help in WSD when the ambiguity is across POSs. For example: Consider the two sentences

1. He chairs the session.
2. The chairs in this room are comfortable.

In the first sentence, the word ‘chairs’ is a verb, and in the second sentence it is a noun. The POS taggers mark the words with appropriate POS tags. In the above examples, if the words are marked with correct POS tags, then the disambiguation is done. The POS taggers use certain heuristic rules, and hence may sometimes go wrong. The reported performances of these POS taggers vary between 95% to 97%. Nevertheless they are still useful, since they reduce the search space for meanings substantially.

Disambiguation in the case of polysemous words requires disambiguation rules. It is not an easy task to frame such rules. It is the context, which plays a crucial role in disambiguation. The context may be

1. the words in proximity, or
2. other words in a sentence that are related to the word to be disambiguated.

The question is how can such rules be made efficiently? To frame disambiguation rules manually would require hundreds of man-years. Is it possible to use machines to automate this process?

The WASP workbench [54] is the best example of how, with the help of a small seed data, machines can learn from the corpus and produce disambiguation rules. Anusaaraka used the WASP workbench to semi-automatically generate these disambiguation rules. The WASP generated rules though are human readable, many a times is just a list of several cases together. As such these rules lack generalisation. Advantage of WASP generated rules is, it is very easy to add, delete and modify the rules. But lack of generalisation explodes the number of rules to a few hundreds for a single word. This makes it un-manageable for a human.

The output produced at this stage is irreversible, since machine makes choices based on heuristics.

4.7.6 Phrase Boundary Marker

A phrase boundary marker is essential to decide the position of the head of the phrase. When one language is head-initial, and the other one is head-final, as in the case of English and Hindi, it is necessary to move the head to the appropriate position. This requires a phrase boundary marker. We provide below an example from English-Hindi pair.

While moving the prepositions from their English positions to the appropriate Hindi positions, a record of their movements is stored, making the transformations reversible. This layer marks the movement by an arrow (\(\rightarrow\)). For example, \(\rightarrow+2\)
Chapter 4: A fresh look at the problem

in table 4.3 indicates that the postposition on is moved to +2 positions in Hindi.

<table>
<thead>
<tr>
<th>He</th>
<th>kept</th>
<th>the</th>
<th>book</th>
<th>on</th>
<th>the</th>
<th>table.</th>
</tr>
</thead>
<tbody>
<tr>
<td>vaha{pu.}</td>
<td>rakha</td>
<td>pustaka</td>
<td>+2</td>
<td>tebala para.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: preposition movement

4.7.7 Target Language Word Order Generation

In this step, proper word order for the TL output is generated. We discuss this module with an example of English-Hindi. Hindi is a free word order language. Therefore, even the anusaaraka output at previous layer makes sense to the Hindi reader. However, this output not being natural in Hindi, may not be enjoyed as much as the one with natural Hindi order. Additionally, it would not be treated as a translation. Therefore in this module an attempt is to generate the correct Hindi word order.

The basic structure of English is SVO. Though Hindi is a relatively free word order language, it is a statistical fact that Hindi has SOV word order. The rules for transforming the English word order into Hindi word order are being worked out. In order to go back, if needed, to English words and English word order, we also maintain a mapping between English word order and Hindi word order.

4.8 Interface for different linguistic tools

The second major contribution of this architecture is the concept of ‘interface’. Machine translation requires language resources such as POS taggers, morphological analyzers, and parsers. More than one kind of each of these tools exist. Hence, it is wise to use these tools. However, there are problems.
For example several parsers for English exists on the internet. Link parser [85], Stanford parser [61], Enju Parser [94], Minipar [59] are some of them.

1. These parsers do not have satisfactory performance. At the most 40% of the time, the first parse is the correct parse. Parse of a sentence tells how the words are related to each other. 90% of such relations in any parse are typically correct.

2. Each of these parsers is based on different grammatical formalism. Hence, the output they produce is also influenced by the theoretical considerations of that grammar formalism.

3. Since the output format for different parsers is different, it is not possible to remove one parser and plug in the other one.

4. One needs a trained manpower to interpret the output produced by these parsers, and to improve the performance of these parsers.

As a machine translation system developer who is interested in the “usable” product one would like to plug-in different parsers and watch the performance. May be one would like to use combinations of them, or may like to do voting among different parsers, and choose the best parse out of them.

The question then is how to achieve it?

It is not enough to have the programs modular. The parser itself is an independent module. What is required is a plug-in facility for different parsers. This is possible provided all the parsers produce an output in some common format. Hence, interfaces are necessary to map the output of parsers to an intermediate form as illustrated in figure 4.8. This interface is based on Paninian Grammar. We discuss the theoretical
We also need a voting program, which decides the best parse among several parses. Such a module then can help one to choose the best parse for a given type of sentence, or may even ‘generate’ altogether a new parse taking the best relations of various parsed outputs. Since no two parsers agree in their output schemes, we tested the voting algorithm on POS taggers, all of which use the same tagset though they differ on formatting details. Figures 4.9 shows the architecture of such a voting machine, and appendix A gives a sample output of such a voting algorithm, run on 5 POS taggers.

4.9 Anusaaraka output and the user interface

Typically languages differ at various levels of encoding viz. morphology, syntax, semantics, etc. Anusaaraka aims at providing ‘faithful’ image of the source language text into the target language. Since the incommensurability between the languages is at various levels, it is not enough to present the ‘gloss’ or śabdasūtra. What is required is a multi dimensional view of the coding – the dimensions referring to various
levels of encoding. To give an analogy, an architect provides various 2-D views of a 3-D building such as plan, elevation, side view etc., similarly we in anusaaraka provide ‘gloss’ of the source language into TL from ‘several views’. The various views are: coding at the word level, word group level encoding, coding at the level of sentence structure, coding at the phrasal level, etc.

We discuss below the requirements of a GUI for anusaaraka with reference to various kinds of information that this interface is supposed to display, and finally with the desired features.

4.9.1 Requirements of an Anusaaraka GUI

• User profile
The anusaaraka GUI should cater to the diverse needs of users. The anusaaraka users may broadly be classified into

— Users who are comfortable in English but face difficulty in constructions such as

Before holding a person responsible for a crime and according
punishment, the motive behind the action must be determined.

For such users, help on words such as *according* should be provided. These users are aware of other regular usages such as ‘according to’, but may not know the meaning of verb ‘accord’.

– Users who are comfortable with simple sentences but face problems with complex verb formations such as ‘let out’, ‘make out’, ‘got off’, etc.

The system should provide help on such complex verb formations, compound words, etc.

– Users with poor knowledge of English with respect to even common syntactic phenomena.

Such users need online lessons of English grammar, with an intelligent user interface providing the necessary help.

– Users who are very poor in English and also weak in analytic skills.

For such users, a layer with Word Sense Disambiguation output and also a phrase boundary marker should provide the necessary help.

– Users who want only ready made translations.

The system should have an access to outputs of various other MT systems, so that in case one MT system fails, other can provide the gist.

**Features**

Users are of two types – intuitive and sensible. The intuitive users would like to understand the system from a holistic view and then would like to take control and operate themselves. The sensible users on the other hand would like the features ‘up front’ or ‘in their face’. Thus the toolbars, dialog boxes etc. are more appropriate for such users. The sensible users, typically outnumber the intuitive users. Also since a tool such as anusaaraka is used by the user only when in need, even intuitive users may prefer to use the dialog box and tool
bars rather than having full control over the system. The developers, on the other hand, would like to have full control over the system.

- **Shortcuts**
  A good interface should also provide ‘intuitive’ shortcuts to avoid the pressing of combinations in a sequence, and deep navigation.

- **Help**
  A help on the usage of interface should be handy.

- **Aesthetics**
  The interface design, the layout, the fonts used, the color combinations should give an aesthetic feeling and relish the user while using the interface.

### 4.9.2 Contents of the Anusaaraka Interface

The content that needs to be displayed is

- Input text in SL
- Translated output in TL
- Intermediate outputs consisting of
  - Notes on the śabdasūtra
  - TL meaning after possible word groupings
  - POS tags
  - Chunk Boundaries
  - Parser outputs
  - Selection of meaning after WSD
- Outputs of various other MT systems
• Help on divergence between SL and TL

• Help on specific but difficult SL constructions

4.9.3 Anusaaraka Interface

Browser based user interface has been developed to display the outputs produced by different layers of anusaaraka engine. The user interface provides a flexibility to control the display. Various dimensions of the user interface are provided by introducing the following:

• frames

• foldable table

• links leading to pop-up windows

• tooltips

• color codes

Figure 4.10 shows the snapshot of sample English-Hindi anusaaraka output. The output is shown in three frames. The top left frame shows the original English text. The bottom frame gives the translation as any machine translations system would give. The top right frame gives a layered output showing the outputs after each of the tasks outlined above. The pop up window helps the user to hide layers in the right top frame.

A short description of each of the layers in the figure 4.10 is given below.

Frame 1

1. Row 1: Original English sentence

2. Row 2: Word level substitution
Chapter 4: A fresh look at the problem

The small rats are killing the big cats in the jungle.

(a) It is the least fragile layer.

(b) Contains Hindi Padasutra (word formula) for each English word.

E.g. small -> *chotā*\(^{\text{alpa}}\)
rats -> *cūhā\(\{s\}\)*

3. Row 3: Word Grouping

The group of words which as a group gives a new meaning are grouped together.

For example in the above sentence,

are + ing = *0_rahā_hai\{ba.\}*

4. Row 4: Word Sense Disambiguation

Attempts to select the appropriate sense according to the context.
For example, the big cats -> $vyāgrādi$

5. Row 5: Phrase Boundary Marker

Different colors mark the phrase boundaries. The movement of prepositions is marked. For example, the Hindi counterpart of ‘in’ is to be moved by ‘+2’ words.

6. Row 6: Hindi Generation at word group level

The prepositions are moved to their correct Hindi positions. Proper forms of the words are generated, taking into account the gender, number, person agreement rules of the Hindi language.

E.g. ‘->meni – jangala’ is changed to ‘— — jangala+meni’.

cūhā{ba.} -> cūhe, etc.

Frame2 : Hindi Translation

1. Proper Hindi sentence is generated.

## 4.10 Anusaaraka: A better approach for Machine Translation

We claim that anusaaraka is a better approach for Machine Translation because it is

1. Robust: It always produces the output. If the machine fails at higher levels, which are in principle fragile, lower level outputs are still available to the user. However to understand the lower level outputs, some training is required.

2. Transparent: The output at different levels makes the whole process of machine Translation transparent to the user. This opens up a new opportunity
for the persons having an aptitude for language analysis to contribute to the Machine Translation efforts even without any formal training in computational linguistics, or NLP.

Table 4.4 describes the differences between anusaaraka and any traditional Machine Translation system.

<table>
<thead>
<tr>
<th>Feature</th>
<th>Typical MT System</th>
<th>Anusaaraka</th>
</tr>
</thead>
<tbody>
<tr>
<td>Goal</td>
<td>Aims to produce Natural translation. In case of failures produce ‘rough’ 4 translation.</td>
<td>To provide access to the Source Language text. The output may be ‘rough’ 5</td>
</tr>
<tr>
<td>Unit of input</td>
<td>Currently, independent sentences</td>
<td>a complete XML document</td>
</tr>
<tr>
<td>System Components</td>
<td>Morph analysers, POS taggers, Parsers, Sense disambiguation modules, Generator</td>
<td>Same as in MT Plus Anusaaraka User Interface</td>
</tr>
<tr>
<td>Sequence of</td>
<td>Outputs are cascaded. So errors too get cascaded</td>
<td>The basic tasks are processed independently. Price paid: Duplication of effort</td>
</tr>
<tr>
<td>Operations</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transparency</td>
<td>Processing is not transparent to the end-user</td>
<td>Processing is transparent to the end-user</td>
</tr>
<tr>
<td>Access</td>
<td>User has access only to the final output</td>
<td>User has access to the output at each level</td>
</tr>
</tbody>
</table>

4what is ‘rough’ is not well defined.

5Here rough is as in the sense of ‘rough journey’ where you are taken to the destination, but the journey is not comfortable.
Chapter 4: A fresh look at the problem

<table>
<thead>
<tr>
<th>Guidelines for Linguists</th>
<th>No specific guidelines</th>
<th>First write an algorithm for ‘Human beings’ and not necessarily for ‘computers’!</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principle</td>
<td>Ad-hoc</td>
<td>“Information Dynamics”</td>
</tr>
<tr>
<td>Approaches</td>
<td></td>
<td>Eclectic: Choose the best of each of these approaches. Use best of the available resources under GPL.</td>
</tr>
<tr>
<td></td>
<td>1. EBMT</td>
<td></td>
</tr>
<tr>
<td></td>
<td>2. Rule based</td>
<td></td>
</tr>
<tr>
<td></td>
<td>3. Statistical</td>
<td></td>
</tr>
<tr>
<td></td>
<td>4. Hybrid</td>
<td></td>
</tr>
</tbody>
</table>
### Consequences

<table>
<thead>
<tr>
<th></th>
<th>1. Later modules are affected by the errors of the previous modules</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Rough is not well defined. Hence users may get mislead.</td>
</tr>
<tr>
<td></td>
<td>3. User can not participate in the development process</td>
</tr>
<tr>
<td></td>
<td>4. Linguists end up in reinventing the wheel again and again.</td>
</tr>
<tr>
<td></td>
<td>1. Parallel processing ensures that different modules do not interfere.</td>
</tr>
<tr>
<td></td>
<td>2. Well defined ‘Roughness’. Theoretically no chances of user getting mislead.</td>
</tr>
<tr>
<td></td>
<td>3. User can participate in the development activity.</td>
</tr>
<tr>
<td></td>
<td>4. Linguist prepares data only once.</td>
</tr>
</tbody>
</table>