Chapter 4

HinMaT Architecture & Lexical Organization

“Design is not just what it looks like and feels like. Design is how it works.”

- Steve Jobs

"A designer knows he has achieved perfection, not when there is nothing left to add, but when there is nothing left to take away."

-Antoine de Saint-Exupery

Having studied the nuts and bolts of MT systems touching different key aspects such as MT basics, its history of development, global and Indian context, prescribed architectures & paradigms and linguistic character of Hindi-Marathi pair, next task for us was to design and develop practically working framework. This entire activity is divided into three phases viz. enumerating the objectives and design philosophy for HinMaT, deriving the system architecture from these objectives, specifying the procedural components by chalkling out lexical data as well as processing tools’ requirements and finally designing and development of the lexical data management and core MT processing tools. This work, which is the major contribution of our research has been divided in two chapters and reported in the current and next chapter. This chapter spells out the design objectives & philosophy of HinMaT framework, approach and detail architecture of the framework followed by the lexical organization and implementation strategy for HinMaT.
4.1 HinMaT objectives and Design philosophy

HinMaT framework is proposed for carrying out Machine Translation of Hindi sentences to Marathi. Considering the time limit and other constraints for doctoral research work, we have scoped HinMaT to translate the simple (kernel) sentences of Hindi to Marathi. Our long term dream is to make HinMaT a full-fledged MT system that would be able to handle all types of sentences and structures. Technical design of HinMaT framework is inspired from ERP software, in which the software is so generic, that the functionality of the software is not constrained to development time business logic/rules, the ERP allow its users to define the business logic rules at any point of time so to say the software is highly configurable. The overall design objectives of HinMaT are based on four important aspects of framework development i.e. basic building blocks i.e. internal modules, process flow, data, and intermediate process data logging for debugging purpose. The overall design objectives of HinMaT are based on scope of research problem as explained in earlier section, formally, they are listed below:

i) HinMaT MT should be able to translate Hindi documents comprising of kernel sentences to Marathi. The basic unit of input for HinMaT translation would be sentence.

ii) The overall design philosophy should be based on the state-of-the-art ‘open factored design’ principle of Software Engineering. So to say design should be highly modular and modules should be operable in plug and play fashion.

iii) The framework design (databases and tools) should be generic so that system functionality should be extendable requiring minimal change in the source code so that it can be applied to other Indian language pairs. Like ERP systems, system functionality should not be confined to only design time business rules. ERP allows its’ users to define the business logic rules at any point of time so that software is highly configurable.

iv) HinMaT should support Unicode encoding.

v) General workflow of HinMaT would consist of Pre-processing, Parsing/Generation and Post-processing stages.
Considering the fact that Direct i.e. word-to-word approach can’t really work well for Hindi-Marathi pair (Bhavsar & Pawar, 19-20 February, 2011), HinMaT should be based on concrete foundation of linguistic theory and Transfer architecture (Analysis-Transfer-Generation) (Vauquois B., 1976).

The ultimate level of analysis in HinMaT should be up to syntactic level with mild semantic information.

Due to common occurrence of gender divergence, Shake and Bake (S & B MT) (Whitelock, 1991), (Whitelock, 23-28 August 1992), (Brew, 1992), (Beaven, 1992) paradigm of MT as discussed in Chapter 2, section 2.2.1.6) should be used as it is more suitable for Hindi-Marathi pair.

HinMaT should make use of grammatical and semantic features during analysis phase for checking the GNP agreement and semantic adequacy of sentence constituents.

HinMaT should use Computational Paninian Grammar (CPG) formalism, as it is the most suitable grammar formalism for representing free word order languages like Indian Languages, as base formalism for analysis.

HinMaT should allow human interaction during Pre-processing and Post-processing stages.

HinMaT should have rich lexical resource base which would cater for selected syntactic as well as semantic features.

Lexical organization should be highly normalized so as to avoid redundancy and inconsistency in data storage, thus promoting optimized data storage.

Lexical organization should be able to handle insertion, deletion and updation of data without causing any inconsistency in the data state.

Document instance specific volatile data such as numbers, time & date, currency, telephone numbers, e-mail ids etc. should be captured at real time and this data should not be stored permanently in regular lexicons.

Data Latency period should be minimum. Actual data access mechanism should rest with RDBMS server software, so that the overall computing work load between front end and back end could be balanced. This can be achieved using ‘stored procedure’ like scripting mechanism.
Lexical database design should be scalable and database management should be easy.

Lexical data should be stored in Unicode format for avoiding the font dependencies and conversion issues.

Overall MT processing time should be optimum.

HinMaT should have debugging support i.e. processing details of intermediate stages should be available for error analysis purpose. Translated sentences with their parse structures should be banked so that they can provide Translation Memory support for HinMaT.

HinMaT should store all processed data such as translated sentences and their parse structures, and update the data usage statistics for Machine learning and future data ranking purposes.

HinMaT should support multiple MT engines and a control mechanism to switch over MT engines.

HinMaT should be robust and fault tolerant. In case of failure to translate complete sentence, partial outputs should be provided.

Allocation of memory and other computing resources at run time should be handled efficiently so that memory management can be done effectively at run time.

Implementation should be done using state-of-the-art s/w development technology.

4.2 Overview of HinMaT Architecture

The HinMaT architecture is inspired from its design objectives. After detail study of Hindi and Marathi languages from computational linguistics (grammar, divergences & processing) perspective and study of various grammar formalisms as presented in Chapter -2, we came to the conclusion that for thorough computational analysis of free word order Indian Languages like Hindi and Marathi, CPG is most suitable formalism. Also as concluded in Chapter 3, Direct (word-to-word) Transfer architecture is not suitable for Hindi-Marathi pair and Interlingua really does not make sense, as it is too advanced for sister pair languages like Hindi and Marathi hence HinMaT MT engine is designed around Transfer architecture. Considering the commonly occurring divergences and their effect on feature agreements of
constituents, in Hindi-Marathi pair, we have chosen Shake & Bake paradigm of MT for HinMaT. In HinMaT process flow, input Hindi sentence is analyzed using CPG parsing at syntactico-semantic level. The parsed analysis is represented using HinMaT’s proprietary notation called ‘HinMaT derivation tree’. This derivation tree is basis of transfer. This derivation tree is used for generating the equivalent target language (TL) sentence in Generator module. Ultimate level of analysis in HinMaT MT is syntactic level. In case of failure of HinMaT engine at the parsing stage, we have also proposed simple Word2Word MT engine based on Direct (lookup translation) approach. This fall back option is invoked by the HinMaT controller upon user’s consent. Actually, HinMaT in full scale perspective is envisioned as multi MT-engine framework. For Present research work, we have restricted ourselves to CPG and transfer based HinMaT MT engine and Word2Word MT engine. Practically speaking, a sentence can’t be just like that translated by MT system and presented to its’ end user. It has to undergo some pre-processing as well as post-processing (editing). Various tasks are performed during Pre-processing and Post-processing stages. Entire translation process in HinMaT takes place in three stages i.e. Pre-processing, core MT system with CPG Parsing/Generation or Word2Word Engine and Post-processing. Each stage of processing requires some lexical resources. HinMaT has full-fledged lexical organization, which is explained in the following section.

**Figure 4.1** presents the birds-eye view of the proposed HinMaT architecture. The detail architecture is explained in the following section.
From above Figure 4.1, it is clear that like any s/w framework, HinMaT architecture also uses mainly two kinds of resources viz. computational tools and lexical resources. The tools can be further classified as primary tools and auxiliary tools. The primary tools include core MT tools and auxiliary (supporting) tools. The later aid the core MT tools during different phases of MT life cycle like Pre-processing and Post-processing. Also to manage lexical resources, we need other set of tools, they are called resource management tools. All tools in HinMaT have been designed and developed from scratch and same are described in the next Chapter (‘Chapter 5 HinMaT Tools’). The core MT engine in HinMaT is based on CPG formalism, in case of failure of this engine at any stage, the Word2Word engine is called. The details about each stage of HinMaT architecture are explained in the following section.

4.3 Detail HinMaT Architecture

As discussed in previous section, HinMaT architecture is based on Transfer architecture with Shake & Bake MT paradigm. MT process flow consists of three stages viz. Pre-processing, CPG Parsing & generation and Post-processing. Each of these stages is explained below.

4.3.1 Pre-processing

Pre-processing is the first stage of MT life cycle. This phase prepares the input document ready for Machine Translation. Tasks that are generally carried out during this phase include word segmentation, clause segmentation, sentence segmentation and extraction, automatic as well as manual phrase marking, NER phrases recognition and marking. The phrase marked sentence is then extracted from the input document and sent to MT engine for translation purpose. Word segmentation and sentence segmentation is required for languages that do not use white space characters like blank space, punctuation markers to separate words and sentences, word and sentence boundaries are marked during segmentation process. Since Hindi as well as Marathi use well defined punctuation markers, these boundaries are clearly marked hence word and sentence segmentation is not required. Also in the present scope clause segmentation is not required as HinMaT operates on only single clause sentences.
4.3.1.1 Phrase Marking

Phrase marking is essential part of MT Pre-processing phase for two reasons, first, all languages make reasonable use of catch phrases or idioms and other such noun phrases. Translation of catch phrases and idioms is known as an open problem, because when a word is employed in a catch phrase/idiom, it may not represent, its exact dictionary meaning (कोशीय अथवा कोशीय). Also the meaning of an idiom or catch phrase may be different from the gross meaning conveyed collectively by its constituent words. TL may have an equivalent catch phrase or idiom conveying same meaning but there may or may not be one-to-one mapping between words of such equivalent SL-TL catch phrases or idioms. E.g. the English catch phrase “do or die” and its Hindi equivalent, “करो या मरो”, here there is one-to-one mapping at word stem level, while another Hindi catch phrase “या को राखे साईयाँ, मार सके  ना कोई” meaning, those who are protected by god can’t be defeated or killed. The Marathi equivalent construction is “देव तारयाला कोण मार”, here there is no one-to-one mapping. Besides these idioms/catch phrases, certain noun phrases like English noun phrases, “Govt. of India” and “Ministry of Home Affairs” due to lexical divergence needs to be translated as “भारत सरकार”, “गृह मंत्रालय” respectively and not as “भारत क सरकार”, “गृह का मंत्रालय”, which are their literal translations. Also as discussed in Chapter-3, translation of verb phrases is complex phenomenon. Secondly, it needs computing power for analysing (parsing) these catch phrases/idioms and noun phrases, when we know that it’s not very helpful. Hence for correct translation of these items, it is wiser to treat them as one word i.e. phrase. When, we further explored the phrase marking aspect, we found that managing exhaustive dictionary of such catch phrases is a tough job, further it may be domain specific, hence phrase marking should be doable in two modes viz. automatic and manual. In automatic phrase marking, the phrases stored in phrasal lexicon are searched in the input text and matched phrases are marked. HinMaT uses hyphen (‘-‘) character to mark phrases i.e. blank spaces within phrase words are replaced with hyphens. e.g. English phrase “Government of India” would be marked as “Government-of-India”. After the automatic phrase marking is done, if the user, from his experience, feels that certain group of words (not present in phrasal lexicon) can be treated as phrase, he should be able to manually select such group and
mark it as phrase. The phrasal lexicon can be augmented with such manually selected phrases during each run by users. We also observed that input text commonly contains certain entities like dates, currency figures, numbers, telephone/mobile phone numbers, zip code, certain type of Id numbers, various registration numbers etc., and list of such items can grow infinitely and it’s not possible to store such entities in the lexicons, because it will be wastage of crucial resource like storage. Moreover these entities are patterned on some predefined format/scheme (pattern). These patterns can be modelled using powerful computing notation called ‘regular expression’ (RE) or ‘finite state machine’ (FSM). Technically such entities are called as ‘Named Entities’ and program which can recognize these entities, is called as ‘Named Entity Recognizer’ (NER). NER entities are to be stored in a temporary lexicon for particular instance of input document/sentence and are flushed after every MT run of that input document/sentence instance. NER phrase marking is done during automatic phrase marking. HinMaT has RE based NER, in which definitions for commonly occurring entities are stored.

4.3.1.2 Sentence Extraction

Phrase marking may be done at document level. After the phrases are marked, the sentences in input document should be extracted for further processing. Sentence boundaries are marked by certain delimiters like dot (‘.’) in most of the languages. Since dot is also used to mark abbreviations like ‘Dr.’ for doctor, ‘Mr.’ for mister etc. hence extracting input document strings merely on the basis of dot will not always lead to correct sentences. Hence, we need to maintain list of exceptional uses of sentence delimiters. This list is stored using Abbreviation Lexicon in HinMaT.

4.3.2 MT Engine

This is the core of HinMaT framework. Phrase marked and sentence extracted document is fed as input to MT engine. Basic unit of input to HinMaT MT engine is a sentence. The sentence flows through Translation MT pipeline as depicted in above Figure 4.1. Sentence is first tokenized by the tokenizer. These tokens are matched in the lexicon and their complete lexical information like POS category, GNP features, and semantic features is fetched. It is common to observe that a word (token) may appear with more than one POS categories, to resolve this POS ambiguity, most NLP
systems use a tool called **POS tagger**. Basically, a POS tagger predicts the correct POS category in the context of input sentence. POS taggers use either statistical methods or shallow parsing techniques. HinMaT does not do any POS tagging rather; it iterates over permutations of POS categories of each token in the input sentence and generates set of *Token Vectors*, one per each permutation and each set i.e. *Token Vector* is tried for translation. Next step is *Chunking*, tokens (words) in each *Token Vector* are grouped together into chunks, on the basis of syntactico-semantic features of CPG parsing. Such chunked *Token Vector* is sent for CPG parsing, where parsing happens mainly in two stages i.e. Intra chunk parsing and Inter chunk parsing. Successfully parsed sentences are represented using derivation tree represented in HinMaT’s proprietary notation. Sometimes parser may produce multiple parses due to structural ambiguity, our parser tries to resolve these types of structural ambiguities using constraint based approach. SL (i.e. Hindi) derivation tree is built and sent to Generator module, which builds TL (i.e. Marathi) derivation tree from SL derivation tree using Transfer Lexicon and Transfer Grammar. Divergences and word reordering are neatly handled by Generator module. In case of divergence, TL side constituents under divergence are parsed. The TL sentence is generated from TL derivation tree and sent for Post-processing stage. If the parsing fails due to incorrect grammatical constructions or other technical reason, control is switched to *Word2Word MT* engine, which simply produces a lookup translation. In either case, if a word is not found, the SL word is transferred to TL in as-it-is form and used in generation process. Such word is appended with ‘_H’ string to indicate that it is SL (Hindi) Word. One of the important features of HinMaT is that syntactic and semantic agreements between the constituents are religiously checked as per the specification of the CPG rule, during both parsing and generation. The detail history of parsing and generation along with time statistics is made available for further analysis and debugging purposes. The same are also stored in the HinMaT database, for Machine Learning purpose. HinMaT UI shows individual sentence’s MT process details in separate TAB view. The exact details of HinMaT UI, Parser and Generator tools are explained in Chapter 5 ‘HinMaT Tools’.
4.3.3 Post-processing

This is the last stage of HinMaT translation, Post-processing does some level post-editing such as synonym selection i.e. lexical ambiguity resolution, Vibhakti symbol resolution etc.

4.3.4 Lexical resources

Since HinMaT is predominantly a Rule based system, it heavily relies on good amount of lexical data. This data is divided into three classes viz. word level lexical data, Rule data and other auxiliary data.

Resources under Word level data are further sub-classified as monolingual and bilingual. Monolingual resources contain Phrasal Lexicon, SL/TL word lexicon, TempNER lexicon and SL/TL verb lexicon, whereas bilingual resources contain Transfer Lexicon for words and verbs. Rule data contains SL/TL CPG grammar rules, Morphology rules (Verb Morphology and Word morphology) and Transfer Grammar.

Entire Lexical organization is designed around the HinMaT POS ontology. Following Figure 4.2 shows this classification with relevant resources under each class. Detail discussion on each resource is presented in following section 4.4, however these resources are introduced briefly in following sub sections.
4.3.4.1 POS Ontology

This is the heart of HinMaT Lexical Organization. Generally NLP researchers follow various POS tag sets like Penn Treebank, IIIT, Hyderabad POS tag set etc. But, in our opinion these POS tag sets give very limited semantic information about the semantic class of word. Semantic class of word is very crucial in Indian Languages computing, because grammar rules in Indian Languages are highly bound to semantic class and that’s the reason why rule based systems fail, if this aspect is not covered properly, hence we decided to merge the syntactic and semantic class information together and devise our own POS ontology, which is three level deep hierarchy, so that one can define coarse as well as fine grained POS tag. More details on POS ontology are discussed in following section 4.4.1.

4.3.4.2 Phrasal Lexicon

This monolingual resource is used to store phrases as explained in earlier section 4.3.1.1. Users can also add phrases at run time during translation process or in offline mode. The structure of this lexicon is flat, it simply stores the phrase and its Language and its numeric Ids.
4.3.4.3 Temporary NER Phrases Lexicon

This Lexicon is used to store NER phrases in the current input document. These phrases are stored temporarily and flushed after every MT run. Like phrasal lexicon, the structure of this lexicon is similar to that of Phrasal lexicon.

4.3.4.4 Abbreviation Lexicon

This Lexicon stores list of abbreviations, typically ending with a dot (‘.’) like punctuation character. This Lexicon is used during sentence extraction process.

4.3.4.5 NER definition Lexicon

All NER definitions are stored in this lexicon. These definitions are used by NER module for marking NER phrases in the input document. Functionality of NER module is scoped to number of NER definitions in this lexicon.

4.3.4.6 SL/TL Lexicon

This lexical resource is backbone of the translation process. It stores detail information about a word such as its POS category, syntactic and semantic features, usage domain etc. We have used the same lexicon to store SL as well as TL words. This lexicon stores all information in compact form. The detail structure of this Lexicon is explained in Lexical organization section 4.4 below.

4.3.4.7 Transfer Lexicon

This lexicon stores SL & TL word mappings. This lexicon also uses flags to record divergences information in SL-TL mapping. This is very crucial lexical resource from MT point of view.

4.3.4.8 SL/TL Verb Lexicon

Since, verbs are of prime importance in CPG and they contain far more information than lexical POS categories, hence HinMaT stores the verb information in the separate lexicon. Besides, verb’s POS in HinMaT ontology, it stores verb’s TAM label, verb agreement information and complete demand frame. A demand frame specifies the
Karaka requirements of verb as prescribed in Paninian analysis. The same lexicon is used to store SL and TL verbs. This entire verb information is stored in compact form.

4.3.4.9 Verb Transfer Lexicon

Like Transfer Lexicon, this lexicon stores SL-TL verb mappings along with agreement divergence and GNP, TAM divergences etc.

4.3.4.10 SL/TL Grammar

The CPG dependency relations are encoded and stored in this lexicon. The rule format specifies POS, GNPC & Semantic agreement constraints on the ‘Head’ and ‘Dependent’ words of dependency relation. Same data table is used to store SL as well as TL dependency rules. The rule also specifies the Head and Dependent words’ relative positions i.e. on which side of the Head word, will the Dependent lay?

4.3.4.11 Transfer Grammar (TG)

Transfer grammar is driving force for HinMaT, as it is based on Transfer architecture. TG stores the mappings between SL and TL rules. Divergences at rule level are also stored. These are further exploited by Generator module for handling divergences.

4.3.4.12 Morphology Rule database

To save the time required for doing morphological analysis, HinMaT encourages storing of entire word morphology in the SL/TL Lexicon in offline mode. For doing so, HinMaT has two morphology generators, one for word morphology and other for verb morphology. Word morphology tool is integrated with SL/TL Lexicon creation tool.

The morphology generation tool operates on morphology rule database, which stores rules for word and verb morphology specifying affixes (both prefixes and suffixes) and instructions for affixation.

4.3.4.13 Tree Bank

This resource stores the complete history of translation cycle for each sentence tried in HinMaT. This history includes, the SL i.e. Hindi sentence, its’ token vector with POS
categories, syntactic (GNPC) and semantic features, SL derivation tree, TL derivation tree, TL sentence, time statistics for parsing, generation and total time etc. This data can be utilized for various analyses as well as machine learning purposes.

For managing all lexical resources as listed above, set of UI programs are required. Hence we have developed such user friendly UI programs with impressive screen layouts to ease data creation and management of each resource. These programs have standard facilities for addition, deletion and editing of the data in the particular lexical resource along with grid view showing all data in that resource.

The following Figure 4.3 shows the detail HinMaT architecture with sub-processes and their associated Lexical resources. The detail working of Parser and Generator are explained in the next chapter.
4.4 HinMaT Lexical Organization

As stated earlier, rich lexical resource is key to success for NLP applications. Hence creating such rich lexical resources and managing them is also an equally important task in NL application development. The lexical organization deals with DBMS aspects i.e. database design, efficient access and management of lexical resources. Schema designing is crucial part of the lexical organization for any NLP application. In our opinion any good lexical organization should have following characteristics:

1. The resource should provide good amount of lexical details about items contained in it, which should include grammatical information as well as semantic information to its’ fullest form.

2. Certain usage characteristics and peculiar features pertaining to word should also be catered and represented so that word sense disambiguation is possible.

3. Drilling down from generalized information (coarse) to specialized information (fine grained) should be possible.

4. Data storage should be language script independent i.e. data should be stored in Unicode format.

5. The data should be stored in the most optimized form, without affecting retrieval time significantly.

6. Data access methodology should rest with RDBMS software and not the front end programs.

The HinMaT’s lexical organization is inspired from these characteristics. Lexical organization is designed around the concept of word. As already explained in Chapter 3, word is a meaningful and morphologically valid sequence of alphabets. A word is basic unit of a sentence. Since HinMaT uses CPG grammar formalism, which is a flavor of dependency grammar (DG), word has greater importance in DG and its parsing. Hence, we chalked out possible information about word and its attributes important from NL computing point of view. Following attributes as listed in Table 4.1 were identified.
Table 4.1 Word Attributes

<table>
<thead>
<tr>
<th>Word Attributes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Morph</td>
<td>Actual word</td>
</tr>
<tr>
<td>Language</td>
<td>Natural Language to which the word belongs to</td>
</tr>
<tr>
<td>Root Word</td>
<td>Stem word of given word</td>
</tr>
<tr>
<td>Word Ending</td>
<td>Word ending (कारा) symbol (vowel) of a word. This symbol (vowel) is very important in Indian languages computing both from morphological and parsing purposes.</td>
</tr>
<tr>
<td>Lexical/POS Category</td>
<td>Lexical Category of the word is basic grammatical information denoting the class/grammatical category of the word.</td>
</tr>
<tr>
<td>Syntactic features</td>
<td>Syntactic features include word’s Gender, Number, Person and Case (collectively called GNPC) and TAM features in case of verbs/auxiliary verbs.</td>
</tr>
</tbody>
</table>

- **Gender**
  - Masculine (M)
  - Feminine (F)
  - Neutral (N)
  - MF
  - FN
  - MN
  - MFN

- **Number**
  - Singular (S)
  - Plural (P)
  - Both (SP)

- **Person**
  - First (F)
  - Second (S)
  - Third (T)
  - FS
  - FT
  - ST

- **Case**
  - Direct
  - Oblique
    - Nominative
    - Accusative
    - Dative
    - Instrumental
    - Ablative
    - Possessive
    - Locative
    - Vocative
So, we can observe that these attributes cater for maximum grammatical (syntactic and semantic) and general information of a word. Lexical category or POS category is basic and very important attribute. We have paid special attention towards Lexical category and have devised special scheme for its representation which is explained in following section.

4.4.1 POS ontology

As explained in section 4.3.4.1 above, the widely followed POS tag sets like Penn Treebank POS tag set, IIIT, Hyderabad POS tag set etc. generally specify grammatical features are applicable to verbs and auxiliary verbs. They are syntactic features of the verb form, they include:

- **Tense**
  - Present
  - Past
  - Future

- **Aspect**
  - Imperfective
    - Inceptive
    - Progressive
    - Ingressive
    - Habitual
  - Perfective

- **Mood(Modality)**
  - Indicative
  - Imperative
  - Interrogative
  - Subjunctive

<table>
<thead>
<tr>
<th>Usage Domain</th>
<th>Specifies the real world domain where this word is specifically applicable. E.g. Official Language, Banking, Tourism, Medical etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic Features</td>
<td>As such number of Semantic Features is not fixed like syntactic features. The list can be open ended, but after deep thought, we have considered following of features:</td>
</tr>
<tr>
<td></td>
<td>• Animate:</td>
</tr>
<tr>
<td></td>
<td>- Animal</td>
</tr>
<tr>
<td></td>
<td>- Human</td>
</tr>
<tr>
<td></td>
<td>- Honorific</td>
</tr>
<tr>
<td></td>
<td>• Locative</td>
</tr>
<tr>
<td></td>
<td>• Abstract/Concrete</td>
</tr>
<tr>
<td></td>
<td>• Instrumental</td>
</tr>
</tbody>
</table>
information like coarse grained and fine grained lexical category, Person-number etc. only. They hardly specify semantic information like semantic class of word. In our observation this semantic class information can be very useful especially in Rule base systems as rules may fail, if applied only on the basis of coarse grained lexical category. Hence, we decided to club this information with basic POS information and thus we have come out with POS ontology which is blend of syntactic as well as semantic class information. The POS sub classification can go up to any level of syntactic and semantic classification. But from the practical implementation view and our detail study on each POS category, we found that three levels deep hierarchy would suffice. Hence in our ontology, POS category is defined using three level deep hierarchy as depicted in the following **Figure 4.4**.

The top tire (level) define the basic lexical category like *Noun, Adjective, verb* etc., while the second tire define sub category under basic lexical category. E.g. *Noun* category is sub-classified as *Proper Noun, Common Noun, and Abstract Noun*. The third and bottom level in the hierarchy define the semantic class under the sub-category. E.g. under Noun-Proper Noun classification, we can have proper nouns for humans, city, village, country, river etc. Same is true for other POS categories like adjective, adverb, pronoun, verb, auxiliary verb etc. We have identified three level deep sub classifications for commonly occurring subclasses of standard POS categories. The entire lexical organization of HinMaT is designed around this hierarchy and word attributes.

<table>
<thead>
<tr>
<th>Lexical Category</th>
<th>Sub-Category</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Noun</td>
<td>03</td>
<td>35</td>
</tr>
<tr>
<td>Adjective</td>
<td>03</td>
<td>12</td>
</tr>
<tr>
<td>Verb</td>
<td>05</td>
<td>08</td>
</tr>
<tr>
<td>Pronoun</td>
<td>07</td>
<td>14</td>
</tr>
<tr>
<td>Adverb</td>
<td>05</td>
<td>01</td>
</tr>
<tr>
<td>Vibhakti</td>
<td>03</td>
<td>23</td>
</tr>
<tr>
<td>Determiner</td>
<td>02</td>
<td>--</td>
</tr>
<tr>
<td>Auxiliary Verb</td>
<td>03</td>
<td>--</td>
</tr>
</tbody>
</table>

**Figure 4.4 HinMaT POS ontology**

**Table 4.2 HinMaT POS Specification Details**
The above table (Table 4.2) summarizes the details of lexical categories at the time of writing this thesis with their sub classification. It is important to note that HinMaT allows extending of this classification as and when required.

Particular category is encoded using tri notation i.e. x.y.z, where ‘x’ is top level lexical category, ‘y’ is its subcategory and ‘z’ is subtype under subcategory ‘y’. The strength of our notation is that, we can use wild card character ‘*’ to mean all types at that level, The power of this notation can be realized and exploited during computation. E.g. POS category specification ‘N.PN.*’ specify all proper nouns under noun category, similarly specification, ‘Adj.*.*’ means all types of adjectives sub categories and all sub types under each adjectival sub category. The later example represents generic class of adjectives, while first one is specific to Proper noun subcategory. So to say our notation gives flexibility to make generic (coarse) as well as specific representation (fine grained). This kind of representation is very powerful in specifying the POS (lexical) categories in the rule base. This POS ontology notation is followed across entire HinMaT framework.

4.4.2 Word level lexical resources

This section describes the lexical Schema design of important lexical resources such as Word Lexicon (both SL and TL) and Verb Lexicon (SL and TL), SL/TL Transfer Lexicon and Verb Transfer Lexicon. The first two resources fall under mono-lingual, while later two come under bilingual resources category. The design philosophy is same for both kinds of lexicons. Both are explained in following section.

4.4.2.1 Word Lexicon

The HinMaT lexical data is stored in various lexicons as shown in Figure 4.1. Amongst these lexicons, SL/TL lexicon and SL/TL verb lexicons are most crucial as they store actual input word data that is used for translation, while other lexicons are storing supporting data. These supporting lexicons are stored using single flat structure, while the word and verb lexicons have been stored using highly normalized format. Also all rule data is stored in using single flat structure for speedy retrieval. Tables are interlinked wherever necessary. Appropriate care is taken while updating the interlinked data, so that the data always remains in consistent state.
Since the cost of storage has come down drastically over last decade, HinMaT stores all possible forms (morphology) of a word/verb in lexicon. This helps to skip the morphological analysis step. The study of word attributes as listed in Table 4.1, suggests that few attributes in this list are cohesive to each other, such cohesive attributes can be grouped together in different groups for optimized design. Our word lexicon design is inspired from the same idea. It is organized in three tables, First table Word_Lexicon stores the basic word information like word, its root word, word ending etc. The second table stores most of the grammatical information associated with word like its Lexical category, GNPC, usage domain etc., the third table stores semantic feature values associated with a word. First-Second and Second-Third tables are connected through common ID. The second table creates the Templates (paradigms) which are shared amongst words. Same is true for third table that stores Semantic Features. Since values stored in each column of second and third table belong to set of predefined values, maximum size of both second and third table can be determined by finding the permutation of possible values of each column, hence this size would cater for virtually infinite number of words in SL and TL. Words in SL/TL would also share the templates. This schema is shown in following Figure 4.5

![Figure 4.5 Word Lexicon Schema Design](image)

4.4.2.2 Verb Lexicon

Since HinMaT uses CPG as the base grammar formalism, hence verb is very crucial lexical resource. We have treated verbs separately in HinMaT because verbs bear additional information i.e. its semantic frame (argument structure) and TAM attributes. TAM denotes TAM features i.e. Tense, Aspect and Mood expressed by the
verb. TAM features control the syntactic structure of semantic frame. The overall lexical information associated with verb is described by Demand frame. The details of demand frame conceptualization are explained in the following section:

4.4.2.2.1 Conceptualization of Verb Frame/Demand Frame

The Lexical information associated with verb can be classified into two classes viz. syntactic information and argument information. The syntactic information includes GNP and TAM information, while argument information specifies semantic as well as syntactic constraints associated with verb form (argument structure). Due to this load of information, abstracting verb is a big challenge. But after giving deep insight to structure of verb, we have devised following structure for verb frame.

Structure of Verb Frame/Demand Frame

Verb Frame is combination of Semantic Frame and Vibhakti Frame. Semantic Frame describes the semantic requirements of the root verb, which depends on argument structure of root verb, while Vibhakti Frame depends on TAM (Tense, Aspect, and Modality) features of each verb form of root verb (Bharati, Chaitanya, & Sangal, 1995). In our implementation, we have considered 09 Karakas26 which are K1-Karta (कता), K2-Karmas (कर्म), K3-Karan (कर्ण), K4-Sampradan (संप्रदान), K5-Apadan (अपदान), K7-Adhikaran (अधिकरण), JK1-Prayojya Karta (प्रयोज्य कता), PK1-Prayojak Karta (प्रयोजक कता) and MK1-Madhyasth Karta (मध्यस्थ कता) (Bharati, et al., 2006). Later three are used with only causative verb forms. The same has be represented pictorially in the following Figure 4.6.

---

26 K6 i.e. रिपधीकारक (Relative) Kakara is also used but it's not part of the verb’s demand frame.
Consider the Morphology Generation of Root Verb – ‘दे’ (give), we can have following verb forms of verb ‘दे’


We can generate these forms by affixing TAM suffixes to the verb root.

E.g. consider the masculine, singular, past tense form ‘दया’, this past tense form can be generated as follows:

\[ \text{दया} = (\text{दे} - \text{◌े}) + \text{ि◌} + \text{या} = (\text{दे} + \text{ि◌}) + \text{या} = \text{दया} \]

The Demand Frame of ‘दया’ is obtained by combining the semantic frame of root verb (‘दे’) and Vibhakti frame of TAM label ‘या’ (yaa) of TAM suffix ‘ि◌’ (‘-इया’-eeyaa) in the following Table 4.3.

---

27 ‘◌’ Unicode representation of ‘(ee) Maatra
<table>
<thead>
<tr>
<th>Demand Frame</th>
<th>K1</th>
<th>K2</th>
<th>K3</th>
<th>K4</th>
<th>K5</th>
<th>K7</th>
<th>JK1</th>
<th>PK1</th>
<th>MK1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Semantic Frame(हा)</td>
<td>+Animate, +Human</td>
<td>+Object</td>
<td>+Human, +Animate</td>
<td>-</td>
<td>+locative</td>
<td>+Human</td>
<td>+Human</td>
<td>+Human</td>
<td></td>
</tr>
<tr>
<td>Vibhakti Frame (इया)</td>
<td>ने</td>
<td>शूया()</td>
<td>ने</td>
<td>को</td>
<td>मै/पर</td>
<td>को</td>
<td>ने</td>
<td>को</td>
<td></td>
</tr>
</tbody>
</table>

Based on these ideas we have designed schema for storing the entire demand frame data. This data is formally stored in verb lexicon. The following Figure 4.6 shows the structure of verb lexicon.

**Figure 4.7 Verb Lexicon Schema Design**

Here Verb Lexicon (VL) and Verb Feature Template Master (VFTM) tables as well as Verb Lexicon (VL) and Verb Demand Frame (VDF) tables are linked through common ID. Karaka details contain constraints on participating noun since the same specification may be applied to different Karakas hence they are stored in separate table. The Demand frame anatomy is explained in following Figure 4.7.

4.4.2.3 Transfer Lexicon

This is yet another important lexical resource that stores the SL-TL word mappings. These mappings are used for actual lexical transfer during the TL generation by Generator module. As such there is many-to-many relationship between SL and TL because one SL word may be mapped with multiple TL words or vice-a-versa. Since
we are storing SL and TL words in the same lexicon, these mappings refer to different rows in the same data table. As discussed in earlier chapter, divergence is common phenomenon that affects the quality of MT output, hence for handling lexical level divergences like POS category divergence, grammatical and semantic feature divergences, these feature level divergences should be captured and stored in the Transfer Lexicon. In our schema design for Transfer lexicon this issue has been catered and we are also storing the feature level divergence information along with SL-TL mapping. The exact structure of HinMaT Transfer Lexicon table is given below in Table 4.4:

**Table 4.4 Transfer Lexicon Schema design**

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Transfer Lexicon Row ID</td>
</tr>
<tr>
<td>SL_ID</td>
<td>SL word’s Lexicon ID</td>
</tr>
<tr>
<td>TL_ID</td>
<td>TL word’s Lexicon ID</td>
</tr>
<tr>
<td>G-divergence flag</td>
<td>This ‘bit’ data field is used to flag the gender divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s gender values are not equal or TL value is generic ‘*’ (divergent).</td>
</tr>
<tr>
<td>N-divergence flag</td>
<td>This ‘bit’ data field is used to flag the number divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s number values are not equal or TL value is generic ‘*’ (divergent).</td>
</tr>
<tr>
<td>P-divergence flag</td>
<td>This ‘bit’ data field is used to flag the person divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s person values are not equal or TL value is generic ‘*’ (divergent).</td>
</tr>
<tr>
<td>C-divergence flag</td>
<td>This ‘bit’ data field is used to flag the Case divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s Case values are not equal or TL value is generic ‘*’ (divergent).</td>
</tr>
<tr>
<td>Anim-divergence flag</td>
<td>This ‘bit’ data field is used to flag the Animate feature divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s Animate values are not equal or TL value is generic ‘*’ (divergent).</td>
</tr>
</tbody>
</table>
whenever the SL and TL word’s *Animate* feature values are not equal (divergent).

**Human-divergence flag**

This ‘bit’ data field is used to flag the *Human* feature divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s *Human* feature values are not equal (divergent).

**Honorific-divergence flag**

This ‘bit’ data field is used to flag the *Honorific* feature divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s *Honorific* feature values are not equal (divergent).

**Locative-divergence flag**

This ‘bit’ data field is used to flag the *Locative* feature divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s Locative feature values are not equal (divergent).

**Instrumental-divergence flag**

This ‘bit’ data field is used to flag the *Instrumental* feature divergence between SL-TL word mappings. The flag is set whenever the SL and TL word’s *Instrumental* feature values are not equal (divergent).

**Category-divergence flag**

This bit data field is used to flag the POS category level divergence between SL-TL word mappings. The flag is set whenever the SL and TL words belong to different POS categories.

### 4.4.2.4 Verb Transfer Lexicon

Since, HinMaT stores the verbs in separate lexicon, SL-TL verb mappings should also be stored in separate transfer lexicon. Verb Transfer Lexicon is responsible for storing these mappings. Like Transfer lexicon, this lexicon stores the mappings between SL and TL verb words along with divergence information. Since verbs contain additional lexical information other than POS categories, divergence information besides GNP and POS category divergences, this lexicon contain, TAM divergences and agreement divergence i.e. agreement shifting from subject (k1) to object (k2) or vice versa. These are stored in separate fields called k1-k2 agreement divergence and k2-k1 agreement divergence fields. TAM divergences refer to difference in Tense, Aspect and Modality
values from SL to TL. Change in either of these hint change in TAM label, which affects the demand frame. Like Transfer Lexicon fields, divergence information is stored using ‘bit’ fields. Description on GNP, POS category divergence is same as 4.4.2.2.

4.4.3 CPG Grammar Resources

Grammar resources are very critical resources as they are the real driving force for all NLP applications. In the context of MT their importance is underlined due to the fact that MT involves two languages. As HinMaT uses CPG formalism as the base formalism for SL analysis and TL generation, hence we designed the schema for encoding CPG grammar constructs (rules). Since CPG belongs to Dependency grammar school, grammar construct (rules) representation is simple and flat in nature, unlike the constituency grammar family formalisms like CFG, LFG, and TAG, HPSG etc. where grammar representation is highly structured and hierarchical at times. Basic construct in the dependency grammar is a labelled dependency relation, which is binary asymmetric relation between two words. CPG needs dependency relations for SL as well as TL analysis and generation respectively. The set of dependency labels is also referred to as ‘dependency tag set’ (pl. see Appendix C). Details on DG and its flavors are already explained in Chapter 3. Hence we need to develop schema for SL and TL CPG grammar as well as Transfer Grammar. These are described in the following sections.

4.4.3.1 CPG SL/TL Grammar

As discussed above basic CPG construct is a dependency relation. A labelled dependency relation is defined over two words of a sentence meeting certain syntactic and semantic criterion/constraints. The syntactic constraints includes POS categories, Dependent word position w.r.t. Head word position, GNPC agreement test, while semantic constraint include semantic agreement test. The label of a dependency relation denotes dependency tag. After giving a thoughtful insight to all these aspects, we have worked out following schema for SL/TL CPG grammar presentation, which is described in following Table 4.5.
<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>CPG dependency Relation ID</td>
</tr>
<tr>
<td>Language</td>
<td>Dependency Relation Language</td>
</tr>
<tr>
<td>Dependency Label</td>
<td>Label of the dependency arc specified in the dependency relation.</td>
</tr>
<tr>
<td>Dependent Word POS Category</td>
<td>Dependent word’s POS category as defined in the HinMaT POS ontology. The use of wild card notation ‘*’ in POS category at any level makes the rule specification very strong as it can be applied to broader class under POS ontology.</td>
</tr>
<tr>
<td>Head Word POS Category</td>
<td>Head word’s POS category as defined in the HinMaT POS ontology.</td>
</tr>
<tr>
<td>Dependent word’s root word’s word Ending Symbol</td>
<td>This field stores the dependent word’s root word’s word ending symbol (कारा कारा). This word ending has significant impact on the syntactic features’ agreement tests between Head word and Dependent word. General observation in Hindi/Marathi is that all ‘आ’ ending (आकारा आकारा) root words will enforce checking of agreement features.</td>
</tr>
<tr>
<td>Head word’s root word’s word Ending Symbol</td>
<td>This field stores the Head word’s root word’s word ending symbol. This word ending has significant impact on the syntactic features’ agreement tests between Head word and Dependent word. General observation in Hindi/Marathi is that all ‘आ’ ending (आकारा आकारा) root words will enforce checking of agreement features.</td>
</tr>
<tr>
<td>Vibhakti Marker</td>
<td>Stores the Vibhakti symbol (post position marker) of the dependent word</td>
</tr>
<tr>
<td>G-Test</td>
<td>This field specifies Test criteria for Gender Agreement between Head word and Dependent word. The Possible values are, ‘Match’, ‘NA’, and ‘MisMatch’ with usual sense.</td>
</tr>
</tbody>
</table>
| N-Test | This field specifies Test criteria for Number Agreement between Head word and Dependent word. The Possible values are, ‘Match’, ‘NA’, and ‘MisMatch’ with usual sense.  

**Match** - Number feature’s values in Head word and Dependent word should match  

**Mismatch** - Number feature’s values in Head word and Dependent word should not match.  

**NA** - The value ‘NA’ means this test is not applicable (required) and hence skipped during dependency testing between pair of words. |
|---|---|
| P-Test flag | This field specifies Test criteria for Person Agreement between Head word and Dependent word. The Possible values are, ‘Match’, ‘NA’, and ‘MisMatch’ with usual sense.  

**Match** - Person feature’s values in Head word and Dependent word should match  

**Mismatch** - Person feature’s values in Head word and Dependent word should not match.  

**NA** - The value ‘NA’ means this test is not applicable (required) and hence skipped during dependency testing between pair of words. |
| C-Test | This field specifies Test criteria for Case Agreement between Head word and Dependent word. The Possible values are, ‘Match’, ‘NA’, and ‘MisMatch’ with usual sense. |

**Match** - Gender feature’s values in Head word and Dependent word should match  

**Mismatch** - Gender feature’s values in Head word and Dependent word should not match.  

**NA** - The value ‘NA’ means this test is not applicable (required) and hence skipped during dependency testing between pair of words.
Semantic Feature Tests

<table>
<thead>
<tr>
<th>Field Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>Transfer Grammar ID</td>
</tr>
<tr>
<td>sLanguage</td>
<td>Source Language of Dependency Relation</td>
</tr>
<tr>
<td>tLanguage</td>
<td>Target Language of Dependency Relation</td>
</tr>
<tr>
<td>sRule_ID</td>
<td>SL Dependency Relation ID</td>
</tr>
</tbody>
</table>

Since HinMaT is based on Shake & Bake paradigm of MT, TL parsing is also performed for cases of divergences. Hence TL rules are also important from MT perspective.

4.4.3.2 CPG Transfer Grammar

Like Transfer Lexicon, Transfer Grammar (TG) is used for mapping SL dependency relations to their TL counterparts. Like other SL/TL Lexicon, same data table is used for representing SL and TL dependency relations. TG is also responsible for storing arc label divergence, and rule precision flag. The exact schema is explained in following Table 4.6.

**Table 4.6 Transfer Grammar schema**

- **Match** - Case feature’s values in Head word and Dependent word should match.
- **Mismatch** - Case feature’s values in Head word and Dependent word should not match.
- **NA** - The value ‘NA’ means this test is not applicable (required) and hence skipped during dependency testing between pair of words.
<table>
<thead>
<tr>
<th>tRule_ID</th>
<th>TL Dependency Relation ID</th>
</tr>
</thead>
</table>
| Side Flip | This field is useful during generation; it records the information regarding change in placement of Dependent w.r.t. to its Head from SL Dependency relation to TL Dependency relation. This field is used for handling the word ordering issues. The value specification is given below:  
*Left2Left*: Dependent lays on left side of its Head word in both SL and TL.  
*Right2Right*: Dependent lays on right side of its Head word in both SL and TL.  
*Left2Right*: Dependent lays on left side of its Head word in SL and on the right side in TL. This is an instance of side flip.  
*Right2Left*: Dependent lays on right side of its Head word in SL and on the left side in TL. This is an instance of side flip.  
*Left2Both*: Dependent lays on the left of its Head word in SL and can lie on either left or right in TL. In such a case generally SL placement i.e. left side placement is applied on TL side.  
*Right2Both*: Dependent lays on the right of its Head word in SL and can lie on either left or right in TL. In such a case generally SL placement i.e. right side placement is applied on TL side. |
| Precision | This field stores the matching preciseness i.e. specific versus generic (N,PN. Per with N.*,* with) w.r.t. the respective POS Categories of Dependent and Head of mapped dependency relations. The value specification is given below:  
*Full*: POS categories match precisely.  
*Partial*: POS categories match partially or either contains generic wild card symbol ‘*’. |
Divergence
This is bit field which is set to ‘true’, if there is divergence in any rule specifications of SL and TL i.e. on Test criterion.

4.4.4 Morphology Rule Resources

It’s a well-known fact that Indian Languages are morphologically rich languages, hence their morphology is complex and extensive. Since morphological analysis step is skipped in HinMaT, all word forms are expected to be present in the lexicon during translation. Hence, we need to store these forms in the lexicon. Keying in all such forms manually is mechanical and hence time consuming as well as boring task. To avoid this situation, we need to devise methodology for automatic generation of complete morphology for a given word (generally root word). For this purpose, we have developed two kinds of Morphology Generators for HinMaT. First one is used for generating morphology of all POS categories except verb, while other generator generates only verb morphology. This is because, as explained in above section on verb lexicon, computationally verbs contain lexically far more information than normal POS category, syntactic and semantic features. Hence the verb morphology generator actually generates the demand frames for the verb besides its morphology.

First type of Morphology generator can generate both inflectional morphology as well as derivational morphology.

Morphological generation in nutshell can be explained as it is a process in which affixing (suffixing as well as prefixing) of affixes is performed on the input word. In this process input word undergoes orthographical changes i.e. it is simple string operation in which few characters are deleted and added to the string. Affixing operation is constrained by various factors such as word ending, ontological POS category, semantic features, GNP features. Based on these requirements, we have designed rule schema for both types of morphology generators, which are explained in the following section.

4.4.4.1 Morphology Rule Schema

The Morphology Rule Schema specifies the detail specification for generating inflectional as well as derivational morphology of input word. The rule specification has two sides input word side and output word (generated morphological word form)
specification. Input word side denotes the qualifying criteria for the rule, while the output word side specifies the GNPC and Semantic features for the generated morphological word form. Rule qualification criterion consists of word attributes such as POS category in HinMaT ontology, word ending symbol/word patterns, and GNPC and Semantic feature specification. Affixes play important role in the morphology generation, number of characters to be removed from input word, affix to be appended to the input word, affix type i.e. prefix or suffix, actual Affix string are important aspects of Affixes. The output word side specifies POS category for morphological form, GNPC and semantic feature specification. Besides these, morphology type i.e. *inflectional* or *derivational*, reference rule Ids for recursive rule application, Valid/Invalid count associated with rule based on total correct and wrong applications of that rule so far, are also stored with the rule. POS category remains same as that of source side in case of inflectional morphology, whereas it changes in case of derivational morphology. The rule structure is given in the following block diagram (Figure 4.9).

<table>
<thead>
<tr>
<th>Primary Rule Information</th>
<th>Auxiliary Information</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input Word Side</strong></td>
<td><strong>Output Word Side</strong></td>
</tr>
<tr>
<td>Rule qualification</td>
<td>Complete specification of generated morphological form</td>
</tr>
<tr>
<td>Criterion</td>
<td>- Output word form after affixation</td>
</tr>
<tr>
<td></td>
<td>- Output word POS category</td>
</tr>
<tr>
<td></td>
<td>- GNPC specification for output word.</td>
</tr>
<tr>
<td></td>
<td>- Semantic feature specification for output word.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>- Input word POS category</td>
<td></td>
</tr>
<tr>
<td>- Input word Ending Symbol</td>
<td></td>
</tr>
<tr>
<td>- Affix Type: Prefix/Suffix</td>
<td></td>
</tr>
<tr>
<td>- Characters to chop from input word</td>
<td></td>
</tr>
<tr>
<td>- Affix</td>
<td></td>
</tr>
<tr>
<td>- GNPC criterion for rule selection</td>
<td></td>
</tr>
<tr>
<td>- Semantic feature criterion</td>
<td></td>
</tr>
<tr>
<td></td>
<td>- Rule ID</td>
</tr>
<tr>
<td></td>
<td>- Morphology Type (Inflectional/ Derivational)</td>
</tr>
<tr>
<td></td>
<td>- Valid Count: total number of correct applications of the rule on input words.</td>
</tr>
<tr>
<td></td>
<td>- Invalid Count: total number of wrong applications of the rule on input words.</td>
</tr>
</tbody>
</table>
4.4.4.2 Verb Morphology Rule & Vibhakti Frame schema

Considering lexically rich information associated with verb, verb morphology can't be thought of in isolation, without considering demand frames. Demand frame is tightly coupled with verb form. Morphology contributes in determining the Vibhakti frame i.e. Vibhakti requirements of verb form for each Karka as explained in Paninian analysis, hence it is associated with verb morphology. Verb Morphology works in similar way as the general morphology as explained in above section 4.4.4.1. Besides GNP features, verb form is also characterized by one more parameter TAM features i.e. Tense, Aspect and Modality expressed by verb form. Affixes that get conjoined with verb form are called as ‘TAM suffixes’. They are inflected on the basis of each of Tense, Aspect and mood suffixes coupled with Gender, Number and Person features. As such verb’s does not have case, but in our opinion the case of verb is expressed by TAM. TAM suffixes are mapped on TAM label (Bharati, Chaitanya, & Sangal, 1995) which can be thought of as root (stem form) of TAM suffixes i.e. TAM suffix minus GNP information. The Vibhakti frame is determined by TAM labels. The verbs also have additional information regarding its’ GNP agreement, which may go with either subject or object or none. The Verb morphology & Vibhakti frame\(^{28}\) schema is given by following Figure 4.10

---

\(^{28}\) Jk1, pk1, mk1- are used in case of Causitive verbs to mark prayojya karta, prayojak karta and madhyasth karta respectively.
4.4.5 Auxiliary Data Resources

This data aid the MT engine, for online and offline evaluation and error analysis process. This data resource is explained below:

4.4.5.1 Process Intermediate data log

This resource records the MT process intermediate logs, processing time statistics etc. Intermediate data stores the output of different stages of the process such as Pre-processing, parsing and generation etc. HinMaT maintains times statistics such as SL parsing time, Pre-processing time and TL generation time. Sentence is a basic unit for such record keeping. This data is crucial for error analysis as well as system performance evaluation purposes.

4.4.5.2 Tree Bank data

HinMaT besides doing MT also stores the translated sentence pairs in this resource. This data can be used in many ways, like it can be used for MT evaluation, secondly it provides Translation Memory to HinMaT for future reference and this data can work as important resource for EBMT and SMT engines. Translation memories are maintained at two levels, sentence level as well as parse tree structure levels. Parsed tree structures of SL and TL are stored using generic notation. This data resource is also used by ‘Tree Viewer’ utility (which is explained in the next chapter), which
shows the SL/TL parse trees visually. One can analyze SL and TL sentence structures for identifying and analyzing divergences.

4.5 Implementation Strategy

The implementation strategy for HinMaT was worked out which is in-line with the HinMaT architecture and design objectives (pl. refer section 4.1 above). Technically HinMaT has three important components viz. Lexical/grammar resources (database and their management), Processing Tools and integrated UI. Following are important features of our strategy:

1. Since the processing architecture of HinMaT is based on ‘Transfer’ architecture with *Shake & Bake* paradigm CPG dependency parsing is performed for source as well as target side, for handling the divergences.
2. Lexical Organization of HinMaT is versatile because it is well grounded syntactically as well as semantically. All lexical resources in HinMaT support Unicode encoding.
3. As entire s/w Design is inspired from ERP like software. In which software functionality is not confined to development time logic, rather users can extend most of the functionality by extending the rule base(s) at run time.
4. Intermediate data of translation process is logged for debugging purpose, along with detail trace for every sentence. This data can be exploited for Machine Learning purpose.
5. HinMaT uses naïve machine learning, while rule application during various stages like morphological generation, this can be easily extended to dependency parsing, and generation.
6. All database operations are handled by the ‘stored procedure’ like mechanisms at backend. This provides security and efficient access.
7. HinMaT is developed on Windows platform using C#.Net®29 and MS-SQL server 2005®.
8. SL parsing takes care of GNPC feature agreement as well as handles the GNP divergence between Hindi-Marathi pair. Semantic features are also used during dependency parsing.

---

29 © Registered trademark of Microsoft Inc., USA.
9. HinMaT design is quite generic so that it can be easily extended to other Indian language pairs with minimum changes in the code.

10. Tools in HinMaT are basic building blocks of the framework and they can be classified as Resource creation tools and Processing tools (all tools are explained in next ‘Chapter 5 HinMaT Tools’). All resources are maintained using resource creation tools.

11. It was decided that resource creation tools would be implemented first and the processing tools later. The processing tools are further sub classified into Pre-processing, Post-processing and core MT tools like parser and generator.

4.6 Summary

In this chapter, we have discussed the most important aspect of HinMaT framework i.e. its detail architecture/building blocks along with lexical organization for HinMaT. HinMaT framework is based on classical ‘Transfer based’ architecture and Shake & Bake paradigm. The process flow in HinMaT includes Pre-processing, Parsing/Generation and Post processing steps. Various tasks performed under each step are also explained. The entire lexical organization of HinMaT is based on three level deep POS ontology, which is blend of pure grammatical and semantic class information. The lexical organization is UNICODE compliant. We have listed various Lexicons and Rule bases required for HinMaT along with their detail database/table structure. The architecture and process flow also helped us to chalk out tool requirement. The next chapter describes various tools that were designed and developed for HinMaT.