Chapter 1

Kathaa Module Packaging and Distribution

By design, the definition of Kathaa Modules is completely decoupled from the actual codebase of the Kathaa Framework. The idea behind completely decoupling module definition from the framework was to facilitate the possibility of a large community of independent and unsupervised contributors and a swarm of community contributed modules which would ultimately be available by a simple to use Kathaa-Package-Manager. While the Kathaa-Package-Manager is not yet implemented, we believe it would be pretty trivial to implement the same because of the way Kathaa Modules are designed.

A set of related Kathaa Modules reside as a Kathaa Module Group in a publicly accessible `git` Repository, and the Kathaa Framework downloads and loads these Modules on the fly just by referencing their publicly accessible `URI` in the Kathaa Framework's configuration file.

1.1 Programmatic definition of a Kathaa Module

The organization of the structure of a Kathaa Module is very similar to how Nodejs modules are organised.Every Kathaa Module requires a file to specify the meta-structure (`package.json`), a file to define the actual computational operation (`index.js`), a file for documenting the module in MarkDown (`description.md`) and any number of other supporting files.

1.1.1 Definition of meta-structure

The meta-structure of a Kathaa Module is always defined in a file called as `package.json`. The basic structure of a `package.json` file is as follows :

```
{
    "name": "module_name",
    "version": "v0.1",
    "icon": "desktop",
    "type": "kathaa-general-module",
    "inports": [
        {
            "name": "input_1",
            "type": "all",
        },
        {
            "name": "input_2",
            "type": "all",
            "optional": true
        }
    ],
    "outports": [
        {
            "name": "output_1",
            "type": "output_type1"
        },
        {
            "name": "output_2",
            "type": "output_type2"
        }
    ],
    "metadata": {
        "label": "Module Label (as it would appear in the Module Library)",
        "other_module_specific_param": "param_value"
    },
    "main": "index.js"
}
```

The `package.json` file has to hold the definition of a javascript object with the following keys :

- `"name"` : The "name" key holds the module name, which has to be unique in the same Kathaa Module Group. In the overall Kathaa Module Library, the name of a particular Module is always namespaced using the name of the Kathaa Module Group
- `"version"` : Holds the module version number
- `"icon"` : Holds the `FontAwesome` icon name for the module, that the module developer would want to be rendered in the Kathaa Visual Interface
- `"type"` : This holds an alphanumeric string referring to the type of the Kathaa Module. The valid values for this parameter are :

  - `"kathaa-general-module"`: (Default Value) This denotes that the Module is a Kathaa General Module as described in Section **??**
  - `"kathaa-blob-adapter"` : This denotes that the Modules is a Kathaa Blob Adapter as described in Section **??**
  - `"kathaa-user-intervention"` : This denotes that the Modules is a Kathaa User Intervention Module as described in Section **??**
  - `"kathaa-resources"` : This denotes that the Modules is a Kathaa Resources Module as described in Section **??**
  - `"kathaa-evaluator"` : This denotes that the Modules is a Kathaa Evaluation Module as described in Section **??**

- `"inports"` : Holds the definition of all the input ports of the module. It is more specifically an array of javascript object definitions for all the input ports. The definition of each of the input ports has to have the following structure :
  - `"name"` : Holds the Input Port name, which has to be a unique alphanumeric string, and will be used to reference the input port in the reset of the Module definition, and also in the Kathaa Visual Interface
  - `"type"` : Holds an alphanumeric string to inform the Kathaa Visual Interface and the Kathaa Orchestrator about the compatibility of a particular Input Port with another Output Port. An Input Port can be connected to another Output Port only if the value of their "type" parameters match. When this parameter is set to `"all"`, it denotes the particular Input Port can be fed using all types of Output ports.
  - `"optional"` : This holds a boolean value stating if the particular Input Port is optional for the Module to go ahead with its defined Computation. The default value of this parameter is `False`. The execution of a Kathaa Graph cannot start until all non-optional Input Ports of all the Modules present in the graph are connected with some Output Port. Kathaa Orchestrator refers to the value of this parameter for dependency resolution during the execution of a graph.

- `"outports"` : Holds the definition of all the output ports of the module. It is more specifically an array of javascript object definitions for all the output ports. The definition of each of the output ports has to have the following structure :
  - `"name"` : Holds the Output Port name, which has to be a unique alphanumeric string, and will be used to reference the output port in the reset of the Module definition, and also in the Kathaa Visual Interface
  - `"type"` : Holds an alphanumeric string to inform the Kathaa Visual Interface and the Kathaa Orchestrator about the compatibility of a particular Output Port with

another Input Port. An Input Port can be connected to another Output Port only if the value of their "type" parameters match.

- `"metadata"`: Holds the module specific metadata and custom parameters which can be changed and edited using the Kathaa Visual Interface, and can be referenced in the definition of the computational operation associated with the module. The "label" sub key is used as the official name of the Module in the Module Library and also in the Kathaa Visual Interface.

- `"main"`: Holds the name of the file which defines the function for the computational operation associated with the module.

1.1.2 Definition of computational operation

The computational operation associated with a particular Kathaa Module has to be defined in a file called `index.js` in the Module directory (The name of the file can ofcourse be changed and instead be configured using the `"main"` key in the `"package.json"` file of the Module.

As mentioned in Section **??**, Kathaa General Modules are the most common and the most powerful types of Kathaa Modules, and their function definition follows the template :

```
module.exports = function (kathaa_input, progress, done){
  // Populate kathaa_output object
  var kathaa_output = {};

  // Populate the individual output-ports for Kathaa-Data-Blob based
  // on the name of the key and the desired computational operation.
  //
  // Lets assume f1(x,y) and f2(x,y)
  // are two functions which have already been defined by us.
  kathaa_output['out_port_1'] = f1( kathaa_input['in_port_1'],
                                   kathaa_input['in_port_2']);
  kathaa_output['out_port_2'] = f2(kathaa_input['in_port_2'],
                                   kathaa_input['in_port_3']);

  // passback computed results via callback
  done && done(null, kathaa_output);
}
```

The key things to note is that, the *kathaa$_i$nput* parameter that is passed as a function is a javascript object, with its keys as the names of all the input-ports as defined in texttttpack-age.json. It represents a single Kathaa Data Blob. The function has to create a new Javascript

Object for the corresponding Kathaa Data Blob that forms the output of the Module. The keys of this new Kathaa Data Blob are expected to be the names of the output ports as defined in `package.json`. The second parameter that is passed is a `progress` object which can be used to inform the Kathaa Orchestrator and the Kathaa Visual Interface of the progress of the computational operation. This parameter would be pretty helpful in case of Modules which deal with a time intensive computational operation, like say training of some Model, etc. The third parameter `done` is a callback object which expects the final Output Kathaa Data Blob at the end of the computational operation.

Similar templates for the other types of Kathaa Modules can be found for reference at :

- `kathaa-blob-adapter` : line-splitter[1]
- `kathaa-evaluation-module` : classification-evaluator[2]

1.1.3 Documentation of the Kathaa Module

The Module directory also needs to have a `"description.md"` file which should ideally hold the Documentation of the Module in `MarkDown`. It is supposed to act more like the "Data Sheet" of the Module to guide new users on how to use the module in the Kathaa Visual Interface, and all of its Input and Output specifications. It is rendered on the Module's page in the Module Library in the Kathaa Visual Interface.

1.2 Defintion of Kathaa Module Groups

Kathaa Module Groups exist as an independent and publicly accessible git repository, with the respective Kathaa Module Directories as sub-folders. In some cases, the developer of a Kathaa Module Group might want to include a few external libraries, in that case they can also define a `libraries.js` file at the root of the repository, which can import libraries in the `GLOBAL` scope while using the name of the Kathaa Module Group as a namespace to avoid conflicts with the external library requirements of other modules. Some definitions of Kathaa Module Groups can be checked at :

- Kathaa Core Modules : https://github.com/kathaa/core-modules
- Sampark Hindi-Urdu Modules : https://github.com/kathaa/hindi-urdu-modules
- Sampark Hindi-Panjabi Modules : https://github.com/kathaa/hindi-panjabi-modules

[1]`https://github.com/kathaa/core-modules/blob/master/line_splitter/index.js`
[2]`https://github.com/kathaa/core-modules/blob/master/classification_evaluator/index.js`

1.2.1 Referencing Kathaa Module Groups in the Kathaa Framework

Kathaa Module Groups can be referenced in the Kathaa Framework by adding them to the texttt"kathaa-module-groups" key in the `"package.json"` of the Kathaa Framework. The key being the intended name of the Kathaa Module Group in the said instance of the Kathaa Framework, and the value being the `URI` to the publicly accessible **git** Repository of the Kathaa Module Group.

1.3 Kathaa Module Services

Most popular NLP Components work in completely different software environments, and hence standardizing the interaction between all of them is a highly challenging task. Kathaa can allow every module to define an optional service by referencing a publicly available *docker container* in the module definition. Kathaa can deal with the life-cycle management of the referenced containers on a configurable set of Host Machines. The corresponding kathaa-modules function definition then acts as a light weight wrapper around this service. This finally enables different research groups to **publish their service** in a consistent and reusable way, such that it fits nicely in the Kathaa Module ecosystem.

1.4 Kathaa Module Library

The Kathaa Module Library loads all the Kathaa Modules from the installed Kathaa Module Groups, and makes them available via the Kathaa Visual Interface for users to mix and match and build Kathaa Graphs. Each Module in the Module Library basically represents a template of the module, and an instance of the Module is created every time a module is dragged onto the Kathaa Graph Editor. In Section 1.5 we list all the modules that are available for users to experiment with using Kathaa, and particularly Section 1.5.2 and 1.5.3 contains numerous Modules from the Sampark Machine Translation Framework[1] which were initially developed by years of work on Sampark [1] by numerous students and researchers associated with the consortium project “Indian language to India Language Machine translation” (ILMT) funded by TDIL program of Dept of IT, Govt. of India.

1.5 Modules currently available in Kathaa Module Library

This section lists the NLP components that are currently available in the Kathaa Module Library as Kathaa Modules.

1.5.1 Kathaa Core Modules

1.5.1.1 sentence_input

- This module is used to feed input into a Kathaa Graph. In the current implementation, it mostly acts as a trigger for Kathaa Orchestrator to ask the user for Input through the Kathaa Visual Interface, and then start "executing" or "resolving" the graph from that Node; but in future implementation, this can play a key role in enabling re-use of Kathaa Graphs as Nodes in designing other Kathaa Graphs.

- URI: `https://github.com/kathaa/core-modules/tree/master/sentence_input`

1.5.1.2 sentence_output

- This module is supposed to hold the actual output from the Kathaa Graph. In the current implementation, it mostly acts as marker for Nodes which might contain the final "results" of the execution of a Kathaa Graph; but in future implementation, this can play a key role in enabling re-use of Kathaa Graphs as Nodes in designing other Kathaa Graphs.

- URI: `https://github.com/kathaa/core-modules/tree/master/sentenceoutput`

1.5.1.3 echo

- A very simple Kathaa Module which simply copies the data in its input ports to its corresponding output ports.

- URI: `https://github.com/kathaa/core-modules/tree/master/echo`

1.5.1.4 custom_module

- An example Kathaa Module which is supposed to act as a starting point for module developers to start building their own Kathaa Modules.

- URI: `https://github.com/kathaa/core-modules/tree/master/custom_module`

1.5.1.5 line_splitter

- A Kathaa Blob Adapter which splits each of its Kathaa Data Blobs into multiple Kathaa Data Blobs by splitting their contents based on line breaks.

- URI: `https://github.com/kathaa/core-modules/tree/master/line_splitter`

1.5.1.6 line_aggregator

- A Kathaa Blob Adapter which is supposed to be a complement to the `line_splitter`, and it combines multiple Kathaa Data Blobs into a single Kathaa Data Blob by concatenating their values using a line break.

- URI: `https://github.com/kathaa/core-modules/tree/master/line_aggregator`

1.5.1.7 generic_aggregator

- A Generic template for Kathaa Blob Adapters which is supposed to act as a starting point for Module Developers to write their custom Kathaa Blob Adapters.

- URI: `https://github.com/kathaa/core-modules/tree/mater/generic_aggregator`

1.5.1.8 ssf_soft_merger

- A very simplified SSF merger which merges two SSF inputs by merging the attributes from the corresponding Nodes in the SSF input. It is a very simple SSF merger and does not account for major structural differences between the two SSF inputs, and makes a very strong assumption about the similarity of the structure of both the SSF inputs. Nevertheless we have been using this in the Kathaa Implementation of the Sampark Hindi-Urdu MT Pipeline and have not faced any major problems yet. The output of this module is a single merged SSF output.

- URI: `https://github.com/kathaa/core-modules/tree/master/ssf_soft_merger`

1.5.1.9 random_merger

- An example merger module, which has multiple inputs and randomly passes on the value of one of its input ports to the output port. This was developed mostly for debugging and demonstration during the development of Kathaa.

- URI: `https://github.com/kathaa/core-modules/tree/master/random_merger`

1.5.1.10 ssf2readable

- This module converts a SSF input into a Human Readable Sentence, by parsing the SSF input, and rendering just the tokens associated with each of the Nodes in the SSF input.

- URI: `https://github.com/kathaa/core-modules/tree/master/ssf2readable`

1.5.1.11 ssf_pos_sequencer

- This module is a sister module to the `ssf2readable` Module, and it also takes a SSF input, and parses it, and finally collects and renders just the POS tags associated with each of the Nodes in the SSF input; and passes it onto the output port.
- URI: `https://github.com/kathaa/core-modules/tree/master/ssf_pos_seque-ncer`

1.5.1.12 resources

- This is an implementation of the Kathaa Resources Module as described in Section **??**. This does not do any processing of the data,(and infact doesnot have input-ports) but instead it stores and provides a corpus of text (or any other data) which can be used by any of the modules in the whole graph during execution.
- URI: `https://github.com/kathaa/core-modules/tree/master/resources`

1.5.1.13 user_intervention

- This is an implementation of the Kathaa User Intervention Module as described in Section **??**. This instructs the Kathaa Orchestrator to halt the execution of a Kathaa Graph at this Node, and lets the user modify the state of the Module, its output, and then it can resume the execution of the graph. This is critical when building Resource creation and annotation workflows, where some corpus has to first go through some form of preprocessing by other modules, and then needs some modification or annotation by a Human, and then it can pass on to any form of post processing by other modules.
- URI: `https://github.com/kathaa/core-modules/tree/master/user_interven-tion`

1.5.1.14 classification_evaluator

- This is an implementation of the Kathaa Evaluation Module as described in Section **??**. This can help create easy to visualize Confusion Matrices to aid in evaluating the performance of any of the subsystems in the Kathaa Graph. Usually this takes the input on one input port from a `Kathaa Resource` module as a reference corpus, and compares it against a similar corpus generated by a subsection of the Kathaa Graph which feeds into its other Input Port.
- URI: `https://github.com/kathaa/core-modules/tree/master/classificati-on_evaluator`

1.5.2 Modules for Sampark Hindi Panjabi MT Pipeline

1.5.2.1 agreementdistribution

- This module processes a SSF input and its main task is to distribute the features of the head of the chunk to its children. The local word grouper computes the features for the head SSF node from various nodes in the local word group and then copies it over to the head. Then the Local word splitter will split the words but it does not assign any features to the words themselves. In summary, the main task of this module is to distribute the features from the head. The possibility to write custom rules for a specific target language is supported, but is not yet implemented in this Kathaa Module.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/agree-mentdistribution`

1.5.2.2 agreementfeature

- The primary task of this module is to change the agreement feature of the nouns according to the target language. There are some nouns like `"paMKA"` which is female in Telugu where as it is male in Hindi. So, for this we have collected list of feminine nouns in the target language(Hindi) and this module marks all the nouns which are in the list as feminine and others as male.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/agree-mentfeature`

1.5.2.3 chunker

- This module is the prediction instance of a Hindi CRF Chunker[4]. Chunking involves identifying simple noun phrases, verb groups, adjectival phrases,and adverbial phrases in a sentence. This involves identifying the boundary of chunks and marking the label. This module simply wraps over a model trained on manually chunked annotated data, predicts the chunks in the SSF input it receives and passes on the output to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/chunker`

1.5.2.4 computehead

- Head computation is the task of computing the heads of noun and verb groups and more importantly to provide sufficient information for further processing of the sentence in accordance with the Paninian Theory. This module performs Head Computation on a SSF input and passes on the results to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/comp-utehead`

1.5.2.5 computevibhakti

- This module computes and adds the Vibhakti of the Noun and Verb groups in the SSF input it receives on it input port. Then it passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/comp-utevibhakti`

1.5.2.6 defaultfeatures

- For any SSF input, the task of this module is to assign the default features to the words which do not have any features. There are some cases where the morph analyzer does not produce features for all the words and also the local word splitter does not provide all the features for the function words it added. So, this module adds default features to such nodes which do not have any features, to enable other modules to treat all the nodes in the SSF input uniformly. The possibility to write custom rules for a specific target language is supported, but is not yet implemented in this Kathaa Module.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/defa-ultfeatures`

1.5.2.7 guessmorph

- In many cases, the Morph Analyzer can produce multiple feature candidates for different SSF Nodes or Node groups. The Guess Morph Module takes a SSF input in its input port, and then tries to remove as many redundant features as it can based on the context of each of the SSF Nodes and Node groups. It finally passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/gues-smorph`

1.5.2.8 pickonemorph

- This module tries to achieve pretty much the same thing as the `guessmorph` Module, but instead of trying to analyze the context of each of the Nodes in the SSF input, it simply picks up the first feature candidate whenever there are multiple feature candidates for any of the SSF Nodes. This is usually used in conjunction with the `guessmorph` module, and it is useful especially in cases where even after the redundant feature elimination by the `guessmorph` module, some Nodes in the SSF input still have multiple feature candidates.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/pick-onemorph`

1.5.2.9 intrachunk

- The task of this module is to handle the intra-chunk agreement. Intra-chunk agreements like Noun-adjective agreement are handled in this module. For example of an Telugu – Hindi MT system, where Telugu is the source language and Hindi is the target language. The Hindi translations of the Telugu phrases (i) 'manchi baludu' and (ii) 'manchi ammayi' would be (i) 'acchA laDakA' and (ii) 'acchI laDakI' respectively. The adjective in Telugu does not carry any gender, number information. However, Hindi adjectives agree in number and gender with the following nouns, (acchA laDakA), (acchI laDakI). This fact of Hindi grammar would be captured by the sentence generator. The task of the Intra Chunk Agreement would be to change the necessary attributes in the feature structure of 'acchA' using rules provided by linguists. This module simply takes a SSF input, modifies it based on the intra-chunk agreement rules for the target language (Panjabi in this case), and passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/intr-achunk`

1.5.2.10 interchunk

- In contrast to the `intrachunk` module, this module deals with the agreement rules and other target language specific syntactic rules between multiple chunks in the SSF input. This module simply takes a SSF input, modifies it based on the inter-chunk agreement rules for the target language(Panjabi in this case), and passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/inte-rchunk`

1.5.2.11 lexicaltransfer

- This module deals with the Lexical Transfer of SSF inputs from Hindi to Panjabi. Lexical Transfer basically refers to translating lexical items from one language to another, based on internal dictionaries for the particular language pair.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/lexi-caltransfer`

1.5.2.12 morph

- This module implements a morph analyzer for Hindi, and it uses internal rules to identify the root and grammatical features of a word. It splits the word into its root and grammatical suffixes, and then passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/morph`

1.5.2.13 parse

- This module is a Simple Parser for Hindi. Parsing is the process of assigning grammatical labels to each chunk/constituent in the sentence. Identification of the grammatical labels (karaka and non-karaka relations) for each word of the sentence helps many applications such as WSD, NER etc. There are a number of approaches, such as rule-based, statistics based, transformation-based etc. which are used for parsing, this module uses a rule based approach[3] based on the Paninian Dependency Framework[2].

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/parse`

1.5.2.14 postagger

- This module implements a CRF based Part of Speech Tagger for Hindi[4]. It takes the SSF input and assign the part of speech tag (e.g. noun, verb, adjective etc.) to each word in the sentence (or each Node in the SSF input). This acts as simple wrapper over a model trained using CRF on manually annotated part of speech data, and while there is a possibility for this module to be extended to support training of models using a custom corpus from a `resource` module, its implementation has been deferred to the future versions of this module.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/postagger`

1.5.2.15 pruning

- Pruning is the task of removing additional abbreviated feature structure information provided by the morph and leaves a most appropriate feature structure based on pos category and case. This module makes sure that the SSF output has one and only feature structure in case of multiple feature structures for any of the Nodes or Chunks in input SSF.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/pruning`

1.5.2.16 root2infinity

- This module modifies the words in a SSF input from its root form to its infinitive form.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/root-2infinity`

1.5.2.17 tokenizer

- This module takes a sentence in raw text, and converts into word level tokens in the SSF format, and passes it along to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/toke-nizer`

1.5.2.18 transfergrammar

- This module deals with transferring the sentence structure in a parsed SSF input of one Indian Language (Hindi in this case) to another Indian Language(Panjabi in this case), and it is a rule based system, which implies that the module uses a set of rules for a particular language pair, and while it is possible to have support for custom rules for this module, it is not yet implemented in this version of the module.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/tran-sfergrammar`

1.5.2.19 transliterate

- This module is a transliteration module for Panjabi and accepts data for Panjabi sentences in SSF format in its input port, and passes on the transliterated SSF data to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/tran-sliterate`

1.5.2.20 utf2wx

- This module converts data in the SSF input from Unicode to WX Notation[2] and passes on the converted data to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/utf2-wx`

1.5.2.21 vibhaktispliter

- This module consumes a SSF input and handles the splitting of vibhakti that was composed in `vibhakti_computation` and substitutes it to the target langauge, and then finally writes the modified SSF to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/vibhaktispliter`

1.5.2.22 wordgenerator

- This module generates words for all the nodes in the SSF input on the basis of the associated feature matrix for that node. It expects the input SSF data in Panjabi.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/wordgenerator`

1.5.2.23 wx2utf

- This module converts data in the SSF input from WX Notation[2] to Unicode and passes on the converted data to its output port.

- URI: `https://github.com/kathaa/hindi-panjabi-modules/tree/master/wx2utf`

1.5.3 Modules for Sampark Hindi Urdu MT Pipeline

1.5.3.1 agreementfeature

- This module processes a SSF input and its main task is to distribute the features of the head of the chunk to its children. The local word grouper computes the features for the head SSF node from various nodes in the local word group and then copies it over to the head. Then the Local word splitter will split the words but it does not assign any features to the words themselves. In summary, the main task of this module is to distribute the features from the head. The possibility to write custom rules for a specific target language is supported, but is not yet implemented in this Kathaa Module.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/agreementfeature`

1.5.3.2 chunker

- This module is the prediction instance of a Hindi CRF Chunker[4]. Chunking involves identifying simple noun phrases, verb groups, adjectival phrases,and adverbial phrases in a sentence. This involves identifying the boundary of chunks and marking the label. This module simply wraps over a model trained on manually chunked annotated data, predicts the chunks in the SSF input it receives and passes on the output to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/chunker`

1.5.3.3 defaultfeatures

- For any SSF input, the task of this module is to assign the default features to the words which do not have any features. There are some cases where the morph analyzer does not produce features for all the words and also the local word splitter does not provide all the features for the function words it added. So, this module adds default features to such nodes which do not have any features, to enable other modules to treat all the nodes in the SSF input uniformly. The possibility to write custom rules for a specific target language is supported, but is not yet implemented in this Kathaa Module.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/defaultfeatures`

1.5.3.4 headcomputation

- Head computation is the task of computing the heads of noun and verb groups and more importantly they provide sufficient information for further processing of the sentence according to the Paninian Theory[2]. This Module takes a SSF input and modifies the feature structures of all noun and verb groups to point to their respective heads, and then writes the modified SSF to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/headcomputation`

1.5.3.5 intrachunk

- The task of this module is to handle the intra-chunk agreement. Intra-chunk agreements like Noun-adjective agreement are handled in this module. For example of an Telugu – Hindi MT system, where Telugu is the source language and Hindi is the target language. The Hindi translations of the Telugu phrases (i) 'manchi baludu' and (ii) 'manchi ammayi' would be (i) 'acchA laDakA' and (ii) 'acchI laDakI' respectively. The adjective in Telugu does not carry any gender, number information. However, Hindi adjectives agree in number and gender with the following nouns, (acchA laDakA), (acchI laDakI). This fact of Hindi grammar would be captured by the sentence generator. The task of the Intra Chunk Agreement would be to change the necessary attributes in the feature structure of 'acchA' using rules provided by linguists. This module simply takes a SSF input, modifies it based on the intra-chunk agreement rules for the target language (Urdu in this case), and passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/intrachunk`

1.5.3.6 interchunk

- In contrast to the `intrachunk` module, this module deals with the agreement rules and other target language specific syntactic rules between multiple chunks in the SSF input. This module simply takes a SSF input, modifies it based on the inter-chunk agreement rules for the target language(Urdu in this case), and passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/interchunk`

1.5.3.7 lexicaltransfer

- This module deals with the Lexical Transfer of SSF inputs from Hindi to Panjabi. Lexical Transfer basically refers to translating lexical items from one language to another, based on internal dictionaries for the particular language pair.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/lexicaltransfer`

1.5.3.8 merger

- This is a SSF merger module which takes the input from a `Named Entity Recognition` module, a Multi Word Expression module and a `chunker` module, and merges the SSF inputs into a single SSF output based on custom rules for handling different structural patterns that might arise based on the expected outputs of the `Named Entity Recognition` module, a Multi Word Expression module and a `chunker` module. This is a much more advanced version of the `ssf_soft_merger`

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/merger`

1.5.3.9 morph

- This module implements a morph analyzer for Hindi, and it uses internal rules to identify the root and grammatical features of a word. It splits the word into its root and grammatical suffixes, and then passes on the modified SSF input to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/morph`

1.5.3.10 multiwordexpr

- This module processes the input SSF to identify and mark contiguous Multi Word Expressions(MWE). This uses a reference dictionary to find the MWEs. For example it will successfully identify and mark the following phrases in a sentence :

  – ABAsa ho conjunct-verb

  – aBinaMxana kara conjunct-verb
  – alaga alaga reduplication

  While there is a possibility to add custom rules to the internal dictionary that is used as a reference; this current implementation does not yet allow users to add custom rules during the execution of the module.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/multiwordexpr`

1.5.3.11 ner

- Named entities are expressions which refer to very specific things for which the referent is a rigid designator. Person names , Organizations (companies, government organisations, committees, etc), Locations (cities, countries, rivers, etc), Date and time expressions etc are considered to be the name entities. This module identifies names within a text and classifies each instance into predefined categories, and then writes the modified SSF to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/ner`

1.5.3.12 pickonemorph

- This module simply picks up the first feature candidate whenever there are multiple feature candidates for any of the SSF Nodes. This is usually used in conjunction with the `guessmorph` module, and it is useful especially in cases where even after the redundant feature elimination by the `guessmorph` module, some Nodes in the SSF input still have multiple feature candidates.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/pickonemorph`

1.5.3.13 postagger

- This module implements a CRF based Part of Speech Tagger for Hindi[4]. It takes the SSF input and assign the part of speech tag (e.g. noun, verb, adjective etc.) to each word in the sentence (or each Node in the SSF input). This acts as simple wrapper over a model trained using CRF on manually annotated part of speech data, and while there is a possibility for this module to be extended to support training of models using a custom corpus from a `resource` module, its implementation has been deferred to the future versions of this module.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/postagger`

1.5.3.14 pruning

- Short One line Description !!

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/pruning`

This module processes a SSF input and its main task is to distribute the features of the head of the chunk to its children. The local word grouper computes the features for the head SSF node from various nodes in the local word group and then copies it over to the head. Then the Local word splitter will split the words but it does not assign any features to the words themselves. In summary, the main task of this module is to distribute the features from the head. The possibility to write custom rules for a specific target language is supported, but is not yet implemented in this Kathaa Module.

1.5.3.15 tokenizer

- Pruning is the task of removing additional abbreviated feature structure information provided by the morph and leaves a most appropriate feature structure based on pos category and case. This module makes sure that the SSF output has one and only feature structure in case of multiple feature structures for any of the Nodes or Chunks in input SSF.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/tokenizer`

1.5.3.16 transliterate

- This module is a transliteration module for Urdu and accepts data for Urdu sentences in SSF format in its input port, and writes the transliterated SSF data to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/transliterate`

1.5.3.17 utf2wx_urd

- This module converts data in the SSF input from Unicode to WX Notation[2] and passes on the converted data to its output port.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/utf2wx_urd`

1.5.3.18 wordgenerator

- This module generates words for all the nodes in the SSF input on the basis of the associated feature matrix for that node. It expects the input SSF data in Urdu.

- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/wordgenerator`

1.5.3.19 wx2utf_urd

- This module converts data in the SSF input from WX Notation[2] to Unicode and passes on the converted data to its output port.
- URI: `https://github.com/kathaa/hindi-urdu-modules/tree/master/wx2utf_urd`

Bibliography

[1] Sampark: Machine translation among indian languages. `http://sampark.iiit.ac.in/sampark/web/index.php/content`, 2016. Accessed: 2016-02-10.

[2] A. Bharati, V. Chaitanya, R. Sangal, and K. Ramakrishnamacharyulu. *Natural language processing: a Paninian perspective.* Prentice-Hall of India New Delhi, 1995.

[3] A. Bharati, M. Gupta, V. Yadav, K. Gali, and D. M. Sharma. Simple parser for indian languages in a dependency framework. In *Proceedings of the Third Linguistic Annotation Workshop*, pages 162–165. Association for Computational Linguistics, 2009.

[4] A. PVS and G. Karthik. Part-of-speech tagging and chunking using conditional random fields and transformation based learning. *Shallow Parsing for South Asian Languages*, 21, 2007.