Kathaa: A Visual Programming Framework for NLP Applications

Sharada Prasanna Mohanty, Nehal J Wani, Manish Srivastava, Dipti Misra Sharma
Language Technology Research Center
International Institute of Information Technology, Hyderabad
{spmohanty, nehal.wani}@research.iiit.ac.in, {m.shrivastava, dipti}@iiit.ac.in

Abstract

In this paper, we present Kathaa[1], an open source web based Visual Programming Framework for NLP applications. It supports design, execution and analysis of complex NLP systems by choosing and visually connecting NLP modules from an already available and easily extensible Module library. It models NLP systems as a Directed Acyclic Graph of optionally parallalized information flow, and lets the user choose and use available modules in their NLP applications irrespective of their technical proficiency. Kathaa exposes a precise Module definition API to allow easy integration of external NLP components (along with their associated services as docker containers), it allows everyone to *publish* their services in a standardized format for everyone else to use it out of the box.

1 Introduction

Natural Language Processing systems are inherently very complex, and their design is heavily tied up with their implementation. There is a huge diversity in the way the individual components of the complex system consume, process and spit out information. Apart from that, many of the components also have associated services which in many cases are really hard to replicate and/or setup. Hence, most researchers end up writing their own in-house methods for gluing the components together, and in some cases, own in-house re-implementations of the individual components, often inefficient re-implementations. And on top of that, most of the popular NLP components make many assumptions about the technical proficiency of the user who will be using those components. All of these factors clubbed together shut many potential users out of the whole ecosystem of NLP systems, and hence many potentially creative applications of these components. With Kathaa, we aim to separate the design and implementation layers of Natural Language Processing systems, and efficiently pack every component into consistent and reusable black-boxes which can be made to interface with each other through an intuitive visual interface, irrespective of the software environment in which the components reside, and irrespective of the technical proficiency of the user using the system. Kathaa builds on top of numerous ideas explored in the academia around Visual Programming Languages in general(Green and Petre, 1996) (Shu, 1988) (Myers, 1990), and also on Visual Programming Languages in the context of NLP (Cunningham et al., 1997).

[1]https://github.com/kathaa/kathaa

2 Kathaa Modules

Kathaa Modules are the basic units of computation in the proposed Visual Programming Framework. They consume the input(s) across multiple input channels, process them, and finally pass on their output(s) across the many output channels they might have. The user has access to a whole array of such modules with different utilities via the Kathaa Module Library. The user can connect together these modules in any combination as he pleases, as long as the modules are compatible with each other. The user also has the ability to tinker with the functionality of a particular module in real time by using an

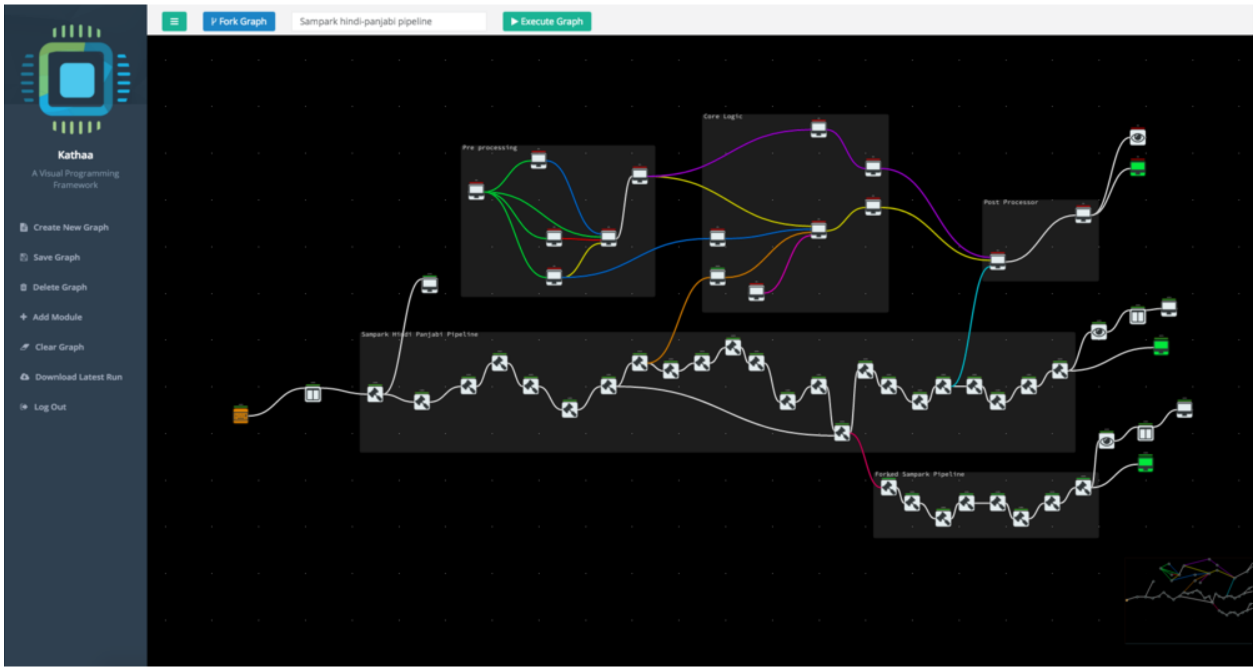

Figure 1: Example of a Hindi-Panjabi Machine Translation System, visually implemented using Kathaa.

embedded code editor in the Kathaa Web Interface during or before the execution of the Kathaa Graph.

2.1 Kathaa Data Blobs

Every module receives inputs across multiple channels, or ports. Every input channel receives the data in the form of a series of **kathaa-data-blobs**, and all the input channels have the exact same number of kathaa-data-blobs. Data blobs are processed in parallel by different instances of the same module during execution, and most modules generate the same number of data blobs across all their output channels. The concept of numerous data blobs spread across multiple input channels enables us to efficiently empower module writers to leverage from the inherent parallelizability in tasks performed by numerous NLP components. For example, some modules might work at the level of sentences, so if we have multiple sentences as inputs to this module, all of them are passed as different kathaa-data-blobs so that the framework can efficiently parallelize their processing depending, of course, on the availability of resources. Similarly, other modules could expect parallelizability at the level of words, or phrases or even a whole discourse. The kathaa-data-blobs were very much inspired by the data-blobs used in Caffe. (Jia et al., 2014)

2.2 Types of Kathaa Modules

2.2.1 Kathaa General Modules

As mentioned previously, most modules produce the exact same number of kathaa-data-blobs as they receive across their input channels. This can be guaranteed because during execution, all the parallel instances of the module are provided only a single kathaa-data-blob in each of their input channels, and when they are done processing, they write a single kathaa-data-blob across their output channels. The *kathaa-orchestrator* deals with the separation of the blobs before passing the inputs to the module instance, and the aggregation of the blobs after each instance of the module has finished processing their corresponding kathaa-data-blobs. These type of modules can be basically called as the **Kathaa General Modules**. To illustrate the above described concepts we implement a very simple Echo module[2], which simply takes in a few data blobs across a single channel, and spits out the same into a single output channel. We also have a very flexible implementation of a Custom Module[3] which can act as a quick starting point when defining Kathaa General Modules.

[2] `https://git.io/vV4RA`
[3] `https://git.io/vV4Rp`

2.2.2 Kathaa Blob Adapters

Kathaa Blob Adapters, on the other hand are a class of Kathaa Modules, which are provided with all the blobs across all their input channels at the same time, and they have the ability to modify the number of blobs and pass it over to their output channels. They can be used in giving the user a more fine grained control over the parallalizability of different parts of their Kathaa graphs by using kathaa-data-blobs. For example, a graph which receives a whole discourse as a single blob, might want to process the sentences parallely, and they could use a line-splitter[4] to split the whole discourse represented as a single kathaa-data-blob into multiple kathaa-data-blobs each representing a single sentence, and when finally the user desired processing of the individual sentences are complete, a line-aggregator[5] could be used to aggregate the processed sentences again into a single kathaa-data-blob. Similar kathaa-blob-adapters could be implemented to deal with splitting and aggregation of kathaa-data-blobs at the level of words, phrases, or even some custom logic. Kathaa Blob Adapters will be crucial in exercising the control over the inherent parallalisation support in Kathaa Orchestrator. For example, in contrast to the example cited above, if we are dealing certain language processing tasks which are inherently not parallalizable after a certain level of granularity, like say Anaphora Resolution, Multi Document Summarisation, etc, the user will have to use an appropriate Kathaa Blob Adapter, to make sure that all the information that is required for the particular language processing task is available as a single blob to be passed onto the module. In the case of Anaphora Resolution, a single Kathaa Blob will contain a string of *N* sentences, and in the case of Multi Document Summarisation, a single Kathaa Blob will contain a string of *M* Documents. In both the previous cases, the module can receive multiple such Kathaa Blobs, which can then be processed parallely based on the availability of resources.

2.2.3 Kathaa User Intervention Module

In some NLP systems, the overall execution of the system might have to halt for some kind of user feedback. Like in the case of resource creation, where for example, you start with a bunch of sentences, parse them using an available parser module, and then you would want to add Anaphora annotations by a human annotator (Sangal and Sharma, 2001). In that case, a **Kathaa User Intervention** Module could be used, where the overall execution at the particular node in the graph halts till the user modifies the kathaa-data-blobs as he pleases and resumes the execution at the said node. Kathaa core implements a Kathaa User Intervention[6] module for reference.

2.2.4 Kathaa Resource Module

Kathaa Resource Modules are the class of Kathaa Modules which do not do any processing of the data, but instead they store and provide a corpus of text which can be used by any of the modules in the whole graph during execution.

2.2.5 Kathaa Evaluation Module

The aim of Kathaa is to provide an intuitive environment for not only prototyping and deployment but also debugging and analysis of NLP system. Hence, we include a class of modules called as **Kathaa Evaluation Modules** which very much like Kathaa Blob Adapters receive all the blobs across all the input channels, and do some analysis and spit out the results into the output channels. While in principle this a subset of Kathaa Blob Adapters, these modules enjoy a separate category among Kathaa Modules because of their utility in designing complex NLP systems. We implement a sample classification evaluator[7] to help researchers quickly come up with easy to visualize confusion matrices to aid them in evaluating the performance of any of their subsystems. This could act as a starting point for easily implementing any other Evaluation modules.

2.3 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 allows every module to define an optional service by referencing a publicly available *docker container* in the module definition. Kathaa deals with the life-cycle management of the referenced containers on a config-

[4]`https://git.io/vV4Rj`
[5]`https://git.io/vV40v`
[6]`https://git.io/vV40U`
[7]`https://git.io/vV40f`

urable 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.

2.4 Kathaa Module Packaging and Distribution

Kathaa Modules reside as a collection of Kathaa Module Groups in a publicly accessible `git` Repository. Each of these modules have a specification definition file called as `package.json`, where the author of the module has to specify the basic meta-data about the module like the name, version, input channels, output channels, etc. The user has the option to reference the corresponding Kathaa-Service by referencing the Docker container in this file. The *type* of the input and output channels can also be specified to mark compatibility of different modules with each other. A sample example of a Kathaa module template can be seen in the case of the custom module[8]. All the supported Module Groups for a particular Kathaa Instance can be referenced directly by their publicly available links on the Kathaa Server, and under the hood, Kathaa deals with the dependency resolution of the modules, downloading of all the modules, instantiation of the associated Docker Container if any, etc.

2.5 Kathaa Interface

Kathaa Interface lets the user design any complex NLP system as a Directed Acyclic Graph with the Kathaa Modules as nodes, and edges being the flow of kathaa-data-blobs between them. Users have the option to not only execute any such graph, but also interact with it in real time by changing both the state and functionality of any of the module right from within the interface. It can be a really useful aid in debugging complex systems, as it lets the User easily visualize and modify the flow of kathaa-data-blobs across the whole Kathaa Graph. Apart from that, it also encourages code-reuse by lettings users "Fork" a graph, or "remix" the designs of NLP systems to come up with better and adapted versions of the same systems.

[8]`https://git.io/vV40J`

2.6 Kathaa Orchestrator

Kathaa Orchestrator is at the core of the whole Visual Programming Framework. Kathaa Orchestrator obtains the structure of the Kathaa Graph and the initial state of the execution initiator modules from the Kathaa Interface, and then it goes on to efficiently orchestrate the execution of the graph depending on the nature and state of the modules, while dealing with process parallelisms, module dependencies, etc under the hood.

3 Use Cases

Kathaa, as a Visual Programming Framework was developed with Sampark Machine Translation System as a use case. We ported all the modules of the Hindi-Panjabi[9] and Hindi-Urdu[10] Translation Pipelines of Sampark Machine Translation System into Kathaa (SAM, 2016). We then demonstrated the use of Kathaa in creation of NLP Resources by the use of Kathaa User Intervention modules, and also moved on to demonstrate visual analysis of different classification approaches by using the Kathaa-Classification-Evaluation module. We are currently also exploring the use of Kathaa in classrooms to help students interact with and design complex NLP systems with a much lower barrier to entry. All these example Kathaa Graphs are the seed Graphs that are included in the repository, and can be used out of the box. It is important to note that these use cases that we managed to explore are only the tip of the iceberg when it comes to what is possible using a framework like Kathaa. One of the key features in Kathaa which enables for it to be used in a whole range of use cases is the easy extensibility. The Kathaa Module Definition API, enables the user of the system to theoretically define any function as a Kathaa Module. Also, Kathaa internally works using event triggers, hence making it a practical possibility to define modules which may run for days or weeks, quite helpful when exploring Kathaa for use cases where the user might want to define a Kathaa Module which trains a model based on some pre-processed data. The NPM(Tilkov and Vinoski, 2010) inspired packag-

[9]`https://github.com/kathaa/hindi-panjabi-modules`

[10]`https://github.com/kathaa/hindi-urdu-modules`

ing system, is again something which we believe can help with large scale adoption of a system like Kathaa. It paves the way for a public contributed repository of NLP components, all of which can be mashed together in any desired combination. The ability to optionally package individual services using Docker Containers also helps make a strong case when pitching for the possibility of a large public contributed repository of NLP components. These are a few things which set Kathaa apart from already existing systems like LAPPS Grid(Ide et al., 2014), ALVEO(Cassidy et al., 2014) where the easy extensibility of the system is a major bottleneck in its large scale adoption. The interoperability between existing systems is also of key importance, and the design of Kathaa accommodates for its easy adaptation to be used along with other similar system. The assumption, of course, is that a wrapper Kathaa Module has to be designed for each target system using the Kathaa Module Definition API. The wrapper modules would be completely decoupled from the Kathaa Core codebase, and hence can be designed and implemented by anyone just like any other Kathaa Module.

A demonstration video of many features and use cases of Kathaa is also available to view at :
https://youtu.be/woK5x0NmrUA

4 Conclusion

We demonstrate an open source web based Visual Programming Framework for NLP Systems, and make it available for everyone to use under a MIT License. We hope our efforts can in some way catalyze more new and creative applications of NLP components, and enables an increased number of researchers to more comfortably tinker with and modify complex NLP Systems.

Acknowledgments

The first real world implementation of a Kathaa Graph was achieved by porting numerous modules from Sampark MT system developed during the "Indian Language to India Language Machine translation" (ILMT) consortium project funded by the TDIL program of Department of Electronics and Information Technology (DeitY), Govt. of India. Kathaa is built with numerous open source tools and libraries, an (almost) exhaustive list of which is available in the Github Repository of the project, and we would like to thank each and every contributor to all those projects.

References

[Cassidy et al.2014] Steve Cassidy, Dominique Estival, Tim Jones, Peter Sefton, Denis Burnham, Jared Burghold, et al. 2014. The alveo virtual laboratory: A web based repository api.

[Cunningham et al.1997] Hamish Cunningham, Kevin Humphreys, Robert Gaizauskas, and Yorick Wilks. 1997. Gate - a general architecture for text engineering. In *Proceedings of the Fifth Conference on Applied Natural Language Processing: Descriptions of System Demonstrations and Videos*, pages 29–30, Washington, DC, USA, March. Association for Computational Linguistics.

[Green and Petre1996] Thomas R. G. Green and Marian Petre. 1996. Usability analysis of visual programming environments: A 'cognitive dimensions' framework. *J. Vis. Lang. Comput.*, 7(2):131–174.

[Ide et al.2014] Nancy Ide, James Pustejovsky, Christopher Cieri, Eric Nyberg, Di Wang, Keith Suderman, Marc Verhagen, and Jonathan Wright. 2014. The language application grid. In *LREC*, pages 22–30.

[Jia et al.2014] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22Nd ACM International Conference on Multimedia*, MM '14, pages 675–678, New York, NY, USA. ACM.

[Myers1990] Brad A. Myers. 1990. Taxonomies of visual programming and program visualization. *J. Vis. Lang. Comput.*, 1(1):97–123.

[SAM2016] 2016. Sampark: Machine translation among indian languages. `http://sampark.iiit.ac.in/sampark/web/index.php/content`. Accessed: 2016-02-10.

[Sangal and Sharma2001] Rajeev Sangal and Dipti Misra Sharma. 2001. Creating language resources for nlp in indian languages 1. background.

[Shu1988] Nan C Shu. 1988. *Visual programming*. Van Nostrand Reinhold.

[Tilkov and Vinoski2010] Stefan Tilkov and Steve Vinoski. 2010. Node. js: Using javascript to build high-performance network programs. *IEEE Internet Computing*, 14(6):80.