2013 13th International Conference on Computational Science and Its Applications

# Enriched Dashboard: An Integration and Visualization Tool for Distributed NLP Systems on Heterogeneous Platforms

Pawan Kumar
Expert Software Consultants Ltd
New Delhi, India
hawahawai@gmail.com

Rashid Ahmad
LTRC, International Institute of Information Technology
Hyderabad, India
rashid.ahmed@research.iiit.ac.in

B. D. Chaudhary
Motilal Nehru National Institute of Technology
Allahabad, India
bdc@mnnit.ac.in

Mukul K. Sinha
Expert Software Consultants Ltd
New Delhi, India
mukulksinha@gmail.com

***Abstract*—Dashboard is an integration, validation, and visualization tool for Natural Language Processing (NLP) applications. It provides infrastructural facilities using which individual NLP modules may be evaluated and refined, and multiple NLP modules may be combined to build a large end-user NLP system. It helps the system integration team to integrate and validate the NLP system. The tool provides a visualization interface that helps the developers to profile (time and memory) each module. It helps researchers to evaluate and compare their module with the earlier versions of same module. The tool promotes reuse of existing NLP modules. Enriched Dashboard supports execution of modules on heterogeneous platforms. It supports execution of modules that are distributed i.e., located on different machines. It has a powerful notation to define the runtime properties of the NLP modules. It provides an easy-to-use graphical interface that is developed using Eclipse RCP. The user can choose a perspective (view) that allows him to perform his task better. Additionally, Eclipse RCP provides a plugin architecture, hence additional end-user functionalities can be easily added to the tool.**

***Keywords-Distributed NLP Tool; Dashboard; Integration and Testing; Visualization; Validation.***

## I. INTRODUCTION

Natural Language Processing (NLP) systems are complex and knowledge intensive. They are usually designed and developed by a team of NLP experts comprising of computational linguists, computer scientists and language/ software engineers.

Unlike conventional software systems, development of NLP systems does not follow a classical software development life cycle (i.e., *specify, design, build, test, and maintain*). NLP systems are composed of many off-the-self re-usable components [18]. NLP system goes through continuous accuracy improvement for considerable long duration, after they are released.

Due to imprecise nature of software specifications for NLP systems and subjective validation criteria, the conventional software development tools are not sufficient for research, development, and delivery of NLP systems.

Historically small group of researchers (or individuals) have worked on a small part of large NLP problems. Their efforts are spatially and temporally distributed with no or little coordination [1]. Large numbers of NLP modules (or components, interchangeably we use the term modules or components) have been built out of these uncoordinated efforts and are available for use. These NLP components are heterogeneous in nature and are with reusability features. To build any large NLP system reusing existing components needs large-scale engineering effort. To build useful end-user NLP systems reusing these existing heterogeneous components, there is an imperative need for integration and testing platform.

This paper presents Enriched Dashboard, a tool for *integration*, *validation* and *visualization* of distributed NLP systems. The tool provides infrastructural facilities using which multiple NLP components of heterogeneous nature may be integrated into a large end-user NLP system; individual NLP components may be evaluated and refined. In addition, the tool helps researchers to time-profile their components as well.

Distinguishing features of enriched Dashboard tool are:

*1) NLP System can be partitioned, and different partitions (comprised of one or more modules) can reside on heterogeneous platforms on separate machines,*

*2) Tool provides powerful notation for describing the runtime properties of individual modules/components as well as the complete NLP system,*

*a) New system modules can be added/removed with easy-to-use user interface, or using declarative notation*

*b) Flexible runtime modes (speed, stepwise, and debug), that provides to run an application in user defined modes*

*c) Supports Conditional run for each module, i.e., at runtime user can skip execution of a module depending on*

978-0-7695-5045-9/13 $26.00 © 2013 IEEE
DOI 10.1109/ICCSA.2013.24

*input parameters (presence or absence of feature structure in ssf input)*

*d) Supports Triggers for modules to promote robustness, e.g., set timeout for a NLP component that goes into an infinite loop*

3) *Visualization interface is developed using Eclipse Rich Client Platform (RCP). Eclipse RCP* [20] provides plugin architecture. Researchers/developers can build useful utility leveraging its plugin architecture.

An elementary version of the tool was earlier released [17]. This Enriched Dashboard has been now released and is being used by consortium members at 11 research institutions. 18 machine translation systems, Sampark MT systems [2, 3], among 9 pairs of Indian languages have been integrated and tested using this tool.

## II. Background: Need for Dashboard Tool

NLP application software development is usually a team effort where the software architecture (its selection or tuning) is done by NLP experts. All the decisions related with language resources, its coverage, rules, corpora, etc., are taken by NLP experts or computational linguists, and algorithms are implemented by language/software engineers. Continuous involvement of NLP experts and computational linguists in the development process is essential as their knowledge cannot be transferred to the software vendors [18]. In addition, the success of the NLP application solely depends upon them as it is essentially their visualization and creation, requiring continuous molding by them during development, and also continuous efforts to improve accuracy after release.

Unlike conventional software development tool, the development environment suitable for NLP applications has to cater, not only the specificity of NLP applications [4], but also to the needs of each type of member of the development team, viz., the NLP experts, the computational linguists and the software engineers. Each one of them have specific requirements from the tool, like, computational linguists would be interested in knowing the *feature structure* produced by a module, while computer scientist would be more interested in time and memory profile of a module.

### A. Specificity of NLP Applications

This subsection describes the salient requirements, emerging due to specificity of NLP applications [4]. These requirements are distinct from conventional software. Some of these requirements may look very similar to conventional software but at times they are significantly different. Hence the requirements that must be covered by the proposed tool are:

- *Modularity*: A set of generic NLP modules have been well identified and developed; or are available off-the-shelf from multiple sources. These modules are re-used for building NLP systems for various domains by proper *adaptation* and *enhancements*. This need requires that the tool must view a system as built by integration of multiple independent modules.
- *Transparency at module interface*: The specifications for most of the NLP systems are approximate. Usually there is no concept of '*correct input*' and a single '*correct output*' [5]. For example, in a machine translation (MT) system grammatically incorrect sentences are valid input as well as valid output. And hence, unlike conventional software, NLP systems' output is not validated against 'correct output', but is validated against criteria specifying threshold of 'accuracy', mostly by human evaluators [6]. Therefore, the tool should provide transparency at module interface level, (i.e., to view input and output of each module whenever required), facilitating the development team to easily isolate the module having a trouble spot, and independently modify it, to improve the overall system.
- *Module Level Flexibility*: In conventional software application, after it is developed and released, the development team comes into picture if and only if there are some residual bugs, and not otherwise. While for an NLP application '*continuous accuracy and performance improvement*' is a generic requirement, and it needs to be continuously improved by NLP experts and computational linguists. Further, the accuracy improvement is done, not by changing program code most often, but by improving/tuning language specific (or language pair specific) data and rules, and also domain specific data/corpora. Hence, the development environment must provide mechanism for easy replacement of a module by its new version, without any repercussion to the remaining set of modules of the system.
- *Heterogeneity of Modules*: NLP applications are complex and computationally intensive. There is no universal theory of language which is universally accepted. Hence there are multiple computational models for language (text) processing. Different NLP module work on different *level of language*, i.e., some work on paragraph level, some on sentence level, some on chunk level, and some on word level. Accordingly, different module would need different data representation model for its efficient implementation. Each module is developed in a specific programming language, the language that provides suitable data representation that suits its computation model. Thus making the implementation far natural, elegant and efficient in that programming language. Sometimes for reasons of reuse off-the-self (source or binary) NLP modules are available in different programming languages. Hence, the integration tool must facilitate integration team to incorporate heterogeneous set of modules in the system.

- *Heterogeneity of Platforms*: Sometimes for reasons of legacy (and also due to unavailability of source code) the NLP modules are forced to run on a specific operating system platform. The integration tool must support execution of modules that are located on heterogeneous platforms, i.e., it must support distributed execution of NLP modules.
- *Time Profiling of Modules*: As the complexity of an NLP application is very high, efficiency has not been a focus for NLP research in general [8]. Hence performance becomes a major concern for software deployment. Therefore, time/memory profiling of each module is an imperative need which an NLP tool must provide so that the development team can concentrate to improve those modules that are time intensive.
- *Robustness of System*: Some NLP components once in a while for a given input, take quite a long time, get into indefinite loop, or even fail. Therefore, robustness at system level becomes another imperative need for any NLP system, and hence, the deployment environment should provide module level timeout facility to terminate the indefinite loop.
- *Resiliency against Module Failure*: As a less precise output is more acceptable than no output. Therefore, the design of the system should be resilient enough to recover a module failure (or its forced termination), and proceed further to give, at least a degraded output.

*Architectural Requirement:* The proposed tool must provide an architectural framework that promotes *reuse* of existing *heterogeneous* NLP components. It should allow integration of modules with *minimum performance penalty*. It should be *resilient against module failures* and provide *graceful degradation* of the NLP system.

To cater to these architectural needs Enriched Dashboard is designed with *Pipelined-Blackboard* architecture. The Blackboard i.e., the *common in-memory data structure* ensures *efficiency* of operation. Pipeline ensures *simplicity* and *flexibility* of integration. Blackboard ensures *resiliency* against module failures, i.e., whenever a NLP module fails to produce any output (after a timeout) the subsequent module continue to operate on in-memory data ensuring graceful degradation for the NLP system.

### B. Objectives of Dashboard Tool

In the light of above mentioned issues we have designed and implemented a software tool that attempts to meets the following objectives:

- Provides Architectural framework for setting up *pipelined-blackboard architecture* system. Able to build a NLP system combining multiple heterogeneous modules (promotes reuse of NLP components), where modules may be *distributed on heterogeneous platforms* [9].
- Supports information interchange (*interoperability*) between broad ranges of modules at highest common level possible using Shakti Standard Format (SSF) [10].
- Supports *integration* of modules written in any programming language either in source or binary form by providing SSF APIs in multiple programming languages.
- Supports module level *Triggers* and *Conditional Run* for each module, which provide robustness to NLP system
- Supports evaluation and refinement of NLP modules/systems via *easy-to-use graphical user interface*.

## III. Related Works

There have been a number of approaches to build software infrastructure for development, testing, and delivery of NLP systems in the past. Some of them provide GUI for manual annotation like General Architecture for Text Engineering [1] and Alembic Workbench [13]. GATE goes beyond the earlier system and provides a component based infrastructure for building Language Engineering (LE) systems [7, 11, 12]. This simplifies the addition of new components to the system and allows components to be configurable, and does not impose any specific standards on the components.

A large number of standalone tools have also been developed. For example, suit of LT Tools [14, 15] for performing tokenization, tagging and chunking on XML text directly. Some of them are designed around a particular technique and some of them do not provide source code for any further extension or adaptation.

Other example of Infrastructure project is ALEP – the Advanced Language Engineering Platform [16]. ALEP in principle is open, but is primarily an advanced system for developing a particular type of data resource, and for doing particular type of tasks.

An elementary tool called Dashboard for integration and testing for NLP systems on single machine has been developed and reported in literature [17].

There have been experimental attempts to build high performance NLP infrastructure by focusing on Generative Programming [8].

Natural Language Tool Kit (NLTK) is a package of NLP components; it provides simple mechanism to build new components using the existing components [19].

## IV. Dashboard Implementation

Enriched Dashboard is an integration and visualization tool for distributed NLP systems on heterogeneous platforms. Dashboard tool has been designed to build NLP systems reusing multiple heterogeneous (or homogeneous)

NLP components. It has a frontend simply called *Dashboard* and a backend called *Dashboard Runtime*. Dashboard provides a graphical interface for setting up and configuring NLP system. Dashboard Runtime performs the tasks of coordination and communication between the modules.

Dashboard provides common representation called *Shakti Standard Format* (SSF), a tree data structure, for storing linguistic information produced by an NLP module in attribute/value pairs called feature structure. The format has both *in-memory* representation as well as *stream* representation. They are inter-convertible using a *Reader* (stream to memory) and *Printer* (memory to stream). The in-memory representation is good in speed of processing, while the stream representation is good for *portability*, *heterogeneous platforms*, and *flexibility*, in general.

It provides a SSF API for accessing in-memory SSF data structure in multiple programming languages. SSF API provides two basic functions, that of *Reader* and *Printer* apart from many other functions for performing basic tree operations on SSF data.

- *Reader* – it reads input data from file/stream and creates an in-memory data structure which can be accessed by modules using SSF API in their programming language.
- *Printer* – it reads in-memory data structure and prints it to a given file/stream in human readable text format

Typically a SSF wrapper is put around a pre-existing module to enable it to access in-memory SSF data structure. Wrappers also encapsulate information about pre-conditions for a module to run. These pre-conditions may be in the form of (data representation, character encoding, level of processing, etc.). Some existing NLP modules have been adapted to operate with SSF. Alternatively, new modules may be developed from scratch using SSF API.

Dashboard acts as a visualization tool for researchers and developers to debug and refine their modules without getting into the finer details of other NLP modules.

### *A. Dashboard – the frontend*

The Dashboard frontend provides a mechanism to setup and configure the NLP System that runs on the Dashboard platform (in our case it is Sampark MT system). It provides following functions:

1) Provision to define NLP *system specifications file* to build a system composed of set of modules in a pipe-line, granularity of pipe can be a single module, set of module comprising a subsystem or complete NLP system. It also provides a mechanism to specify common API being used by one or more modules of the system. In case of Sampark MT system it is SSF API.

2) Provision to define each module's *form properties* (form in which module is available for reuse/integration). Form properties would include parameters like,

- *Module Type (form)* – module available as an *executable* like a compiled C program; or as a *subroutine* written in Perl/C; or as a *Class file* or *jar file* written in Java; or as module/executable-program resident on *remote machine*
- *Programming language – Perl, C, Java, Python, or Shell,* more programming languages can be added later
- *Level of processing – word, sentence or chunk*,
- *Libraries and other include-folders required for system integration,*

3) Provision to define each module's *runtime properties*, that include parameters like,

- *Input type* – STREAM, FILE, MEMORY, CONSOLE
- *Output type* – STREAM, FILE, MEMORY, CONSOLE
- *Dependency* – module name(s),
- *Data formats* – like SSF, BIO, TNT in case of Sampark MT system
- *Character encoding for modules* – UTF-8, WX-Roman,
- *Conditional execution of a module* – conditional execution means module would get executed only if specific feature structure is available in the input data, currently limited to availability of single feature structure in input data
- *Run mode* – SPEED, STEPWISE, or DEBUG,
- *Breakpoints* – Y/N (for interactive usage), once a breakpoints is enabled after a module, Enriched Dashboard stops execution of subsequent modules. It begins execution only after the user intervenes.
- *Triggers for the module* – currently used for failure handling of module by setting a time out.

4) Provision to compile a given NLP systems' specifications file, and generate an executable *Runtime*. This Runtime is eventually executed to run the configured NLP system.

*5) Provision to edit systems specifications file for reconfiguring the system, mainly to replace old version of the module with a newer version.*

### *B. Dashboard Runtime*

Dashboard Runtime performs the tasks of coordination and communications between the modules. It performs following tasks:

- Runs the modules in a pre-specified sequence as configured in the systems specifications file,
- Applies runtime wrappers for handling inter-module *data formats* and *character encodings*,
- Applies *Reader* and *Printer* functions as and when required by two interfacing modules,
- Schedules execution of modules on same machine or located on different machines,
- Applies Triggers (timeout) to modules,
- Produces intermediate input/output in DEBUG mode.

### C. Dashboard as a Visualization Tool

Dashboard provides visualization and debugging functionality to researchers and developers. It performs following tasks:

- Display of intermediate input/output at module level, subsystem level, and system level, mainly to analyze accuracy of the system,
- *Step-by-step* module run – interactively at module level,
- *Here-and-now debugging tool* – manually change intermediate output and run the system further. It would help to find multiple bugs in single execution. It also helps in integration and system testing,
- *Save a session* – save system and intermediate input, output of each module and also of complete system for post analysis of module/system by the development team,
- *Time Profile* – Time profile for each sentence translation at every module level
- *Support Tools* – provides multiple support tools for transliteration, character encoding conversion among Indian languages, format validation tool,
- *User Defined Plugins* – User defined plugins can be added to enhance utility of the tool. Accuracy calculator for 2 NLP modules currently available as Eclipse RCP Plugin. More could be added with a little effort.

## V. SAMPARK MT SYSTEM: AN APPLICATION USING DASHBOARD

In this section we illustrate the power and flexibility of Dashboard tool. We provide here screen shots of Punjabi-Hindi Sampark MT system (*sampark-pan2hin*). Figure 1, shows the normal view of the Dashboard. For reasons of clarity, we have labeled the icons in Figure 1, with numbers in red. On the Tool bar (labeled-2) we have *Run* button (labeled-1), and *View* button (labeled-6) on extreme right. Below the Tool bar on extreme left is the *Application Explorer* pane (labeled-8). It shows the list of Dashboard applications (*sampark-pan2hin* system and *sampark-hin2pan* system). Just below that, i.e., *sampark-pan2hin* system all the modules of *sampark-pan2hin* system is listed. On the right of this pane we have a tabbed window where input text (labeled-3), in Punjabi language, *punjabi.txt* (visible), and output text (labeled-4), in Hindi language, *punjabi-output.txt* (this tab is invisible) are shown. Below this pane input/output from a module, *lexical-transfer* is shown (labeled-9 and labeled-10). Label-11 shows the error log if any from the system. Label-12 shows the selected module name along with selected sentence number. Label-13 shows the time profile for the selected module. Label -15 and 16 shows tool bar for switching views of input and output data useful to computational linguists. The user can select views from a list of views i.e., SSF or XML or Native.

Figure 2, shows complete translation of Punjabi text into Hindi. Left pane shows the input text composed of 10 sentences, written in source language Punjabi, and the right pane shows translated output text, as the output from Sampark system written in target language Hindi.

Once the system is executed completely, the session can be saved for future analysis. In case the user wants to analyze the intermediate output of any specific module for a specific sentence, he can do so through Dashboard. For this he has to first expand the module list in the Application Explorer pane. Select the module whose intermediate output he wants to analyze. Once the user selects the module, he gets a Dashboard view as shown in Figure 3.

Figure 3, shows input to lexical-transfer module in the left pane, and output from lexical-transfer module into the right pane for sentence number 5 in SSF format. Time taken to execution of lexical-transfer module is shown at the bottom of the pane earlier labeled-13 in Figure 1.

To start any application system has to be configured, which was done chronologically earlier but is being explained now. Figure 4, shows system setup and configuration view. In the Application Explorer pane user has to add an application first. Once he adds an application, *DashboardSpec.xml* icon appears in the Application Explorer pane. When the user selects *DashboardSpecs.xml* icon, a tab window earlier labeled-5 becomes visible, 'Applications Module Specification' pane is opened. On the bottom of this pane there are 5 tabs, viz., *Overview*, *Global Prop.*, *Runtime*, *Modules*, and *DashboardSpec.xml*. *Overview* tabs gives an overview about system configuration and setup. '*Global Prop*' tab configures the global properties for Dashboard Runtime, like, location of SSF API for various languages. *Runtime* allows passing runtime commands to the Dashboard (like UNIX command line parameters). *Modules* tab allows defining runtime properties of each module. Figure 4, shows the runtime properties of Punjabi Morph Analyzer. In the figure, only standard properties are defined. In the *DashboardSpec.xml* tab, user can manually edit the DashboardSpec.xml file.

Figure 5, also shows the system setup and configuration view. But here we show how runtime properties for a distributed module that is residing on a remote machine is being configured. For a remote module, for example *parser*, we need to define module-type as *remote*. For module-path we need to define *username*, *hostname*, and *absolute-module-path* on a remote machine. As it is shown in Figure 5, value for module-path is

*"expert@10.2.8.84:/home/parser/pan/parser_pan.pl".*

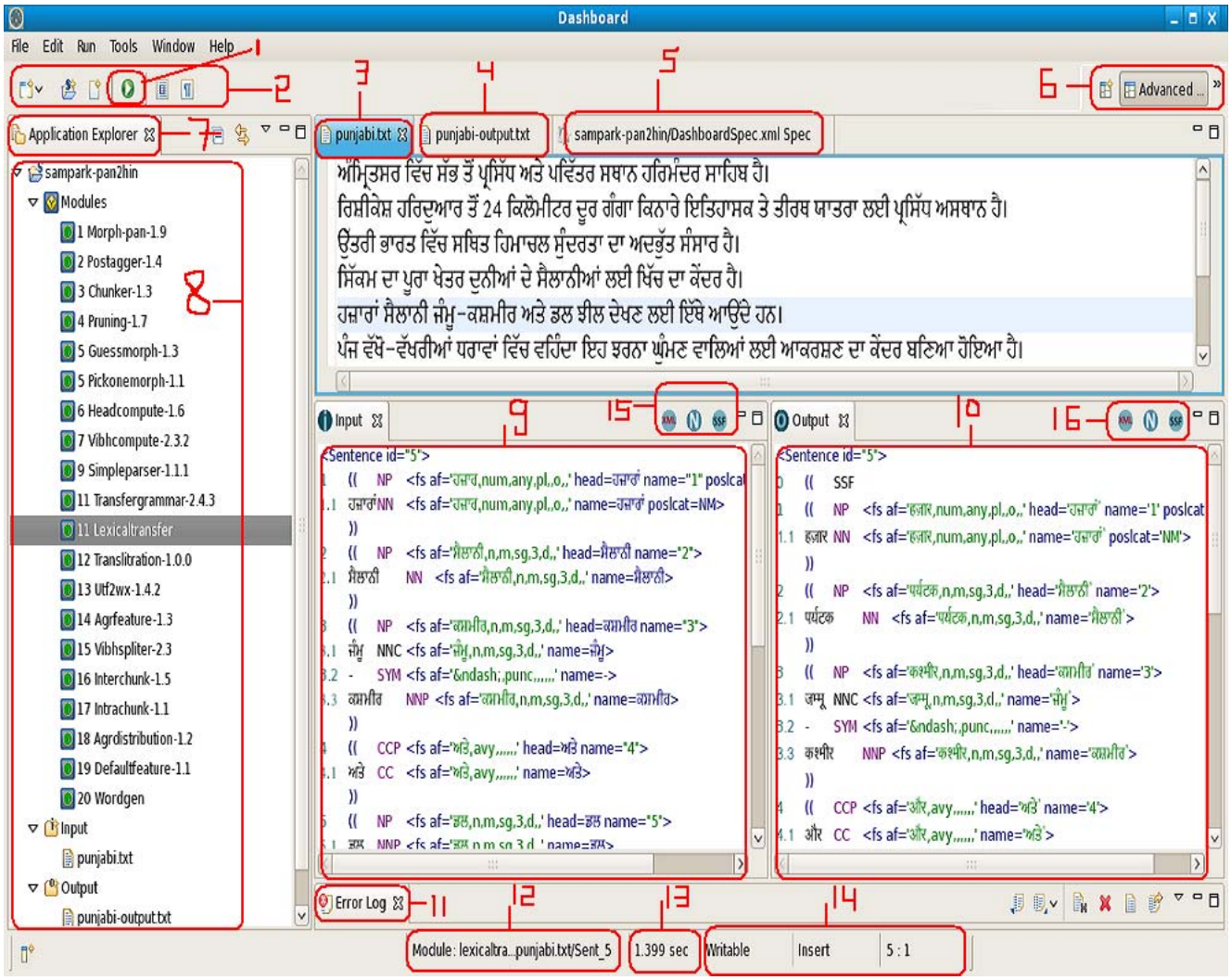


Figure 1: Shows the normal view of Dashboard Tool, icons/panes manually labeled for describing the tool functionality.

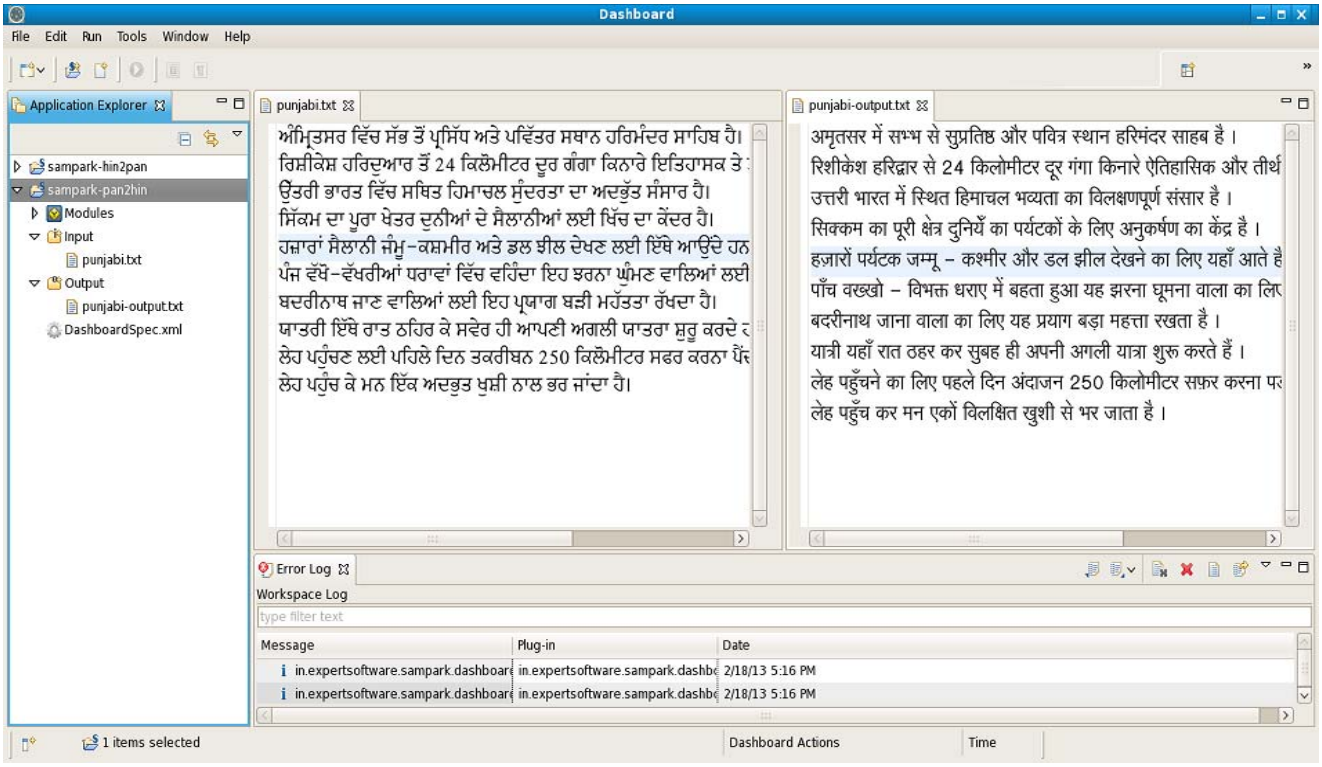


Figure 2: Shows the Punjab-Hindi MT system input and output text for 10 sentences.

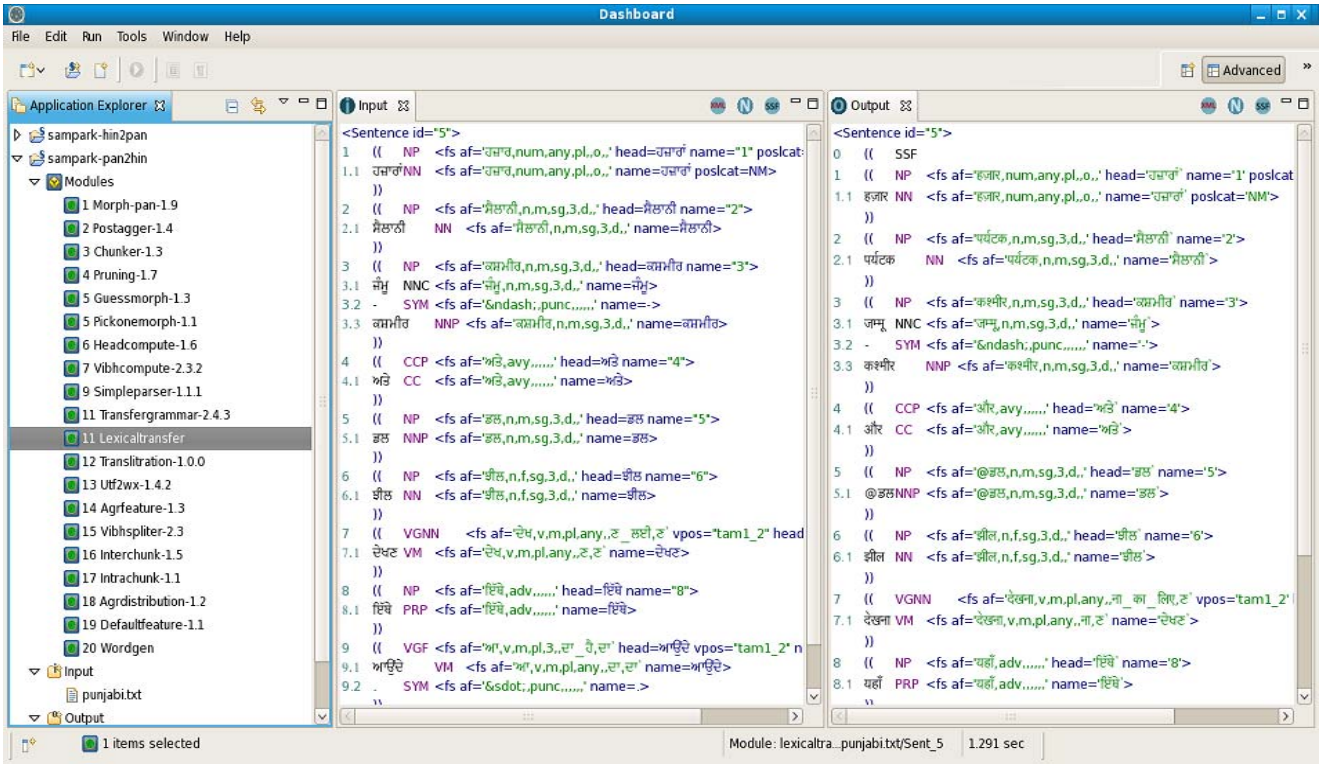


Figure 3: Shows Punjabi-Hindi MT systems' module level input and output data in SSF format for lexical-transfer module for sentence number 5.

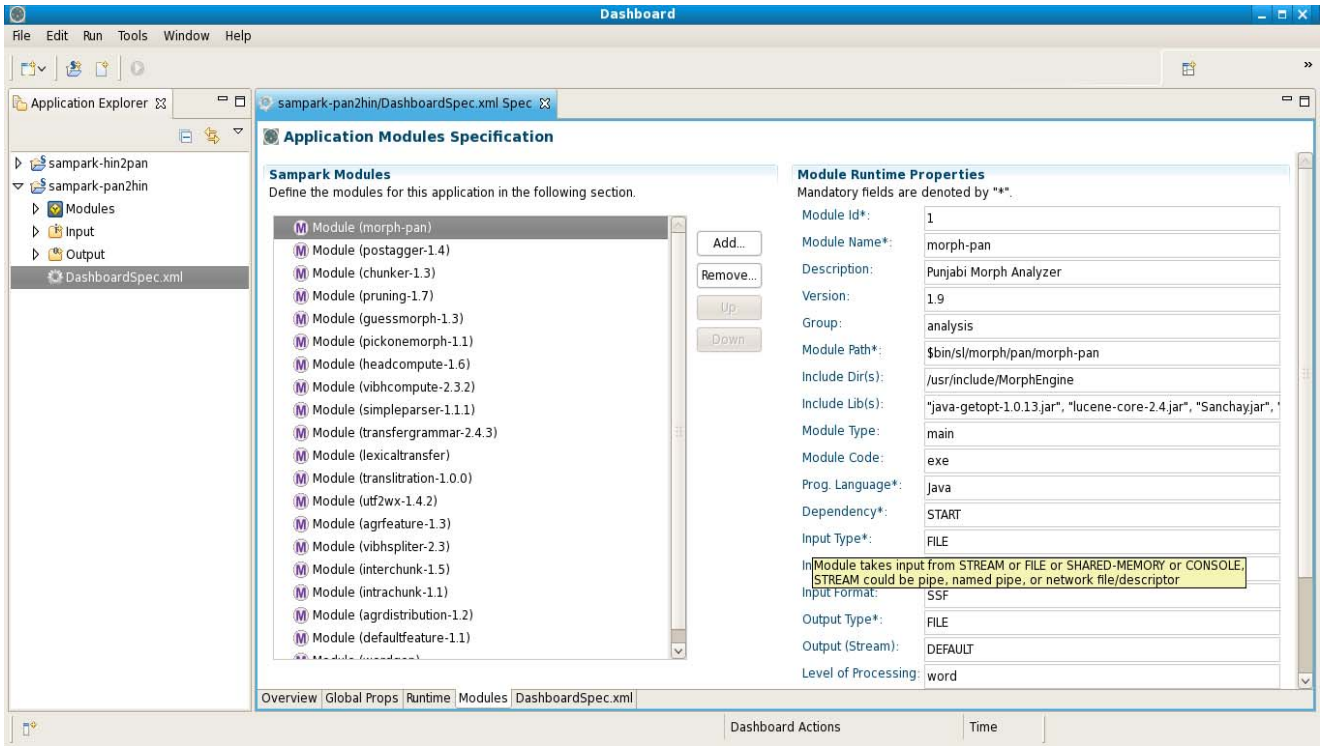


Figure 4: Shows Application setup and configuration view. Right hand side of widow shows the runtime properties for Punjabi Morph Analyzer. It also shows help text, if the mouse is hovered on the text labels

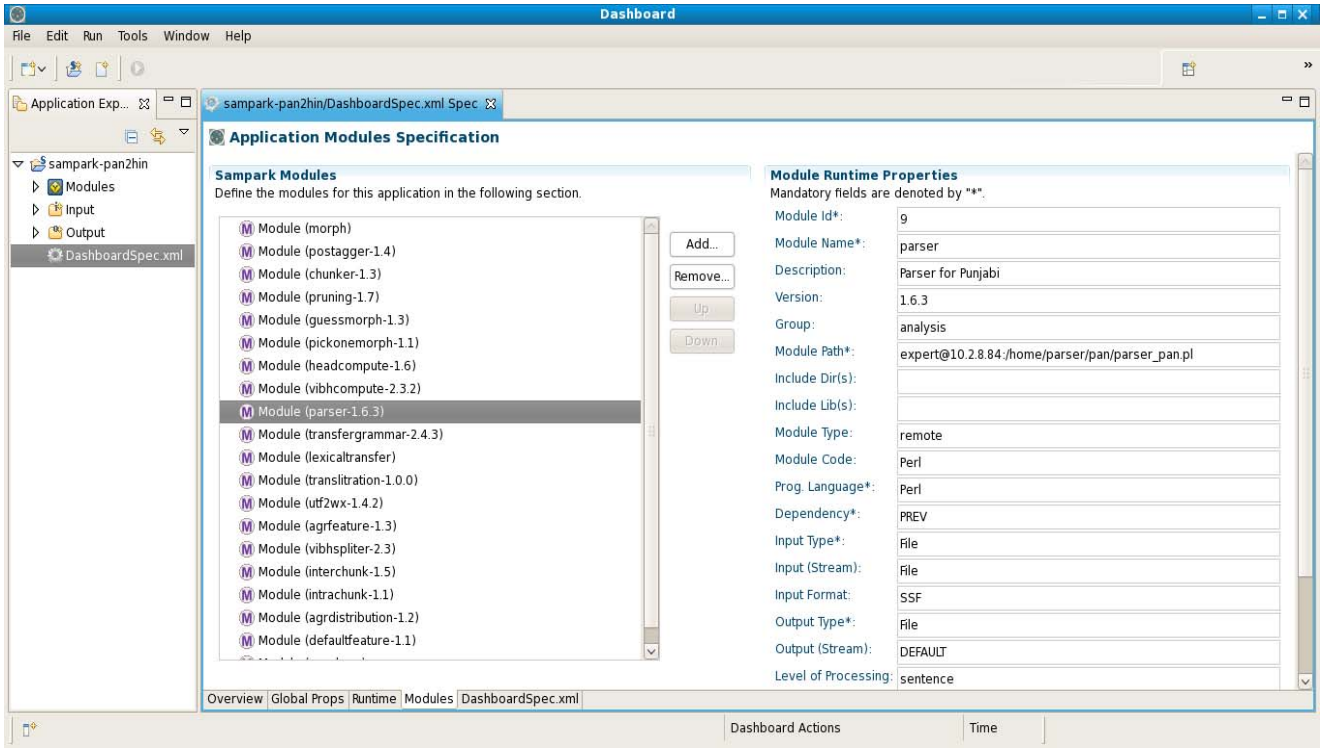


Figure 5: Shows Application setup and configuration view configuring runtime properties of a remote module.

## VI. Current Usage of Enriched Dashboard

The Enriched dashboard is being used at 11 research institutions. More than 100 users (comprising of researchers, computational linguists, and software developers) have been using this tool for a year now. 18 Sampark MT systems have been integrated and tested using this tool in 9 Indian language pairs. Each MT system comprised of 15-20 modules depending upon the language pair.

Enriched Dashboard allows a developer to integrate his module with the rest of the system and test it, without getting into the details of other modules. It helps him evaluate and compare his module with the earlier versions of his module.

In the Sampark MT system some of the modules are highly compute intensive like, Hindi parser, so for performance reason they are deployed on another machine. Similarly dictionary size of synset dictionary (called *lexical-transfer*) can run into 4-5 millions words. Using current version of Dashboard such modules that require large compute resources or large memory for efficient computation can remain deployed on separate machines. These characteristics of NLP systems are being effectively handled by the distributed feature of the Enriched Dashboard.

The visualization interface of the Dashboard is built using Eclipse RPC. It is written in Java. The backend of Dashboard is written in Perl. At present size of source code of backend and frontend combined is approx. 26,000 lines of code.

We are continuously getting feedback and improving the tool for improving the end-user experience.

## VII. Conclusion and Future Work

This Enriched Dashboard tool provides infrastructural facilities to build large NLP systems reusing heterogeneous (or homogeneous) NLP modules. It supports execution of modules that are distributed on heterogeneous platforms. It provides powerful notation for specifying the runtime properties of modules comprising the NLP system. It provides easy-to-use graphical interface for setting up and configuring the NLP system.

This Enriched Dashboard tool is built on Eclipse RCP plugin architecture, and is available for Linux platform. So the user of the tool has to be familiar with the Linux platform. We intend to enhance this tool, so that its user interface becomes browser based. In that case it would be used by more users due to its simplicity of usage.

## Acknowledgment

We sincerely thank TDIL group, Dept. of Information Technology, Govt. of India in granting and supporting such a challenging project. All NLP researchers and software engineers of the participating institutions (viz., IIIT Hyderabad, IIT Mumbai, IIT Kharagpur, Central University Hyderabad, AU-KBC Research Center Chennai, CDAC NOIDA, Jadavpur University, Kolkata, IIIT Allahabad, and IISc Bangalore) need special thanks who have been using this tool and have helped in identifying new requirements for Dashboard tool. Their continuous feedback has not only improved its functionality, but also stabilized it as a product.

We would sincerely thank our colleagues, Rambabu, Phani, Avinash, and Sanket without whose tireless effort current version of the tool would not be possible.

We would sincerely thank Prof. Rajeev Sangal, to allow us to work on such innovative project because 'Dashboard as a NLP Tool' was primarily his idea.

## References

[1] H. Cunningham, K. Humphreys, R. Gaizauskas, and Y. Wilks, "Software Infrastructure for Natural Language Processing," Proceedings of the fifth Conference on Applied Natural Language Processing, pp. 237-244, 1997.

[2] Sampark: Machine Translation System among Indian languages, http://sampark.org.in, last accessed on 15-Feb-2013.

[3] G. Anthes, "Automated Translation of Indian Languages," CACM, vol. 53 (1), pp. 24-26, 2010.

[4] J.L. Leidner, "Current Issues in Software Engineering for Natural Language Processing," Proceedings of the HLT-NAACL workshop on software engineering and architecture of language technology systems, vol. 8, pp. 45-50, 2003.

[5] EAGLES, Expert Advisory Group on Language Engineering, "Evaluation of Natural Language Processing Systems (Final Report)," DG XIII of European Commission, 1996.

[6] R. Moona, S. Singh, R. Sangal, and D.M. Sharma, "MTeval: An Evaluation Methodology for Machine Translation Systems," Proceedings of SIMPLE-04: Symposium on Indian Morphology, Phonology and Language Engineering, IIT Kharagpur, 2004.

[7] Hamish Cunningham , Yorick Wilks , Robert J. Gaizauskas, "New Methods, Current Trends and Software Infrastructure for NLP", 1996.

[8] James R. Curran, "Blueprint for a high performance NLP infrastructure," Proceedings of the HLT-NAACL workshop on Software engineering and architecture of language technology systems, vol. 8, pp 39-44, 2003.

[9] R. Sangal, "Architecture of Shakti Machine Translation System," IIIT Hyderabad, 2004.

[10] R. Sangal: Dashboard: A Framework for Setting Blackboards, IIIT Hyderabad, 2005.

[11] H. Cunningham: Software Architecture for Language Engineering, Ph.D. Thesis, University of Sheffield, U.K., June 2000.

[12] Diana Maynard and Valentin Tablan and Hamish Cunningham, et al., "Architectural Elements of Language Engineering Robustness," Journal of Natural Language Engineering, vol. 8 (3), pp. 257-274, June 2002.

[13] David Day, John Aberdeen, Lynette Hirschman, Robyn Kozierok, Patricia Robinson, and Marc Vilain, "Mixed-initiative development of language processing systems," Proceedings of the fifth conference on Applied natural language processing, pp. 348-355, 1997.

[14] Andrei Mikheev, Claire Grover, and Marc Moens, "Xml tools and architecture for named entity recognition," Journal of Markup Languages: Theory and Practice 1, 3:89–113, 1999.

[15] Claire Grover, Colin Matheson, Andrei Mikheev, and Marc Moens. 2000. LT TTT - a flexible tokenisation tool. Proceedings of Second International Language Resources and Evaluation Conference, pages 1147–1154, Athens, Greece, 31 May – 2 June.

[16] N. K. Simkins, 1994, an Open Architecture for Language Engineering. In First Language Engineering Convention, Paris.

[17] Pawan Kumar, Arun K. Rathaur, Rashid Ahmad, Mukul K. Sinha, and Rajeev Sangal, "Dashboard: An integration and testing platform based on blackboard architecture for NLP applications," Proceedings of the sixth IEEE International Conference on Natural Language Procssing and Knowledge Engineering, pp. 506-513, Beijing, China, 2010.

[18] Pawan Kumar, Rashid Ahmad, Arun K. Rathaur, Mukul. K. Sinha, Rajeev Sangal, "Reengineering Machine Translation Systems through Symbiotic Approach", Proceedings of 3rd International Conference in Contemporary Computing, part 2, pp. 193-204, NOIDA, India, August 9-11, 2010.

[19] Edward Loper, Steven Bird, 2002, NLTK: The Natural Language Toolkit. In Proceedings of the ACL-02 Workshop on Effective tools and methodologies for teaching natural language processing and computational linguistics – vol. 1, pp. 63-70

[20] Jeff McAffer, Jean-Michel Lemieux, and Chris Aniszczyk, "Eclipse Rich Client Platform", Second Edition, Addison Wesley.