Dashboard: An Integration and Testing Platform based on Blackboard Architecture for NLP Applications

Pawan Kumar*, Arun Kumar Rathaur*, Rashid Ahmad*, Mukul K Sinha*, and Rajeev Sangal**
*Expert Software Consultants Ltd., NOIDA, India **International Institute of Information Technology, Hyderabad, India

*Abstract-***The paper presents a software integration, testing and visualization tool, called Dashboard, which is based on pipe-lined blackboard architecture for family of natural language processing (NLP) applications. The Dashboard helps in testing of a module in isolation, facilitating the training and tuning of a module, integration and testing of a set of heterogeneous modules, and building and testing of complete integrated system as well. It is also equipped with a user-friendly visualization tool to build, test, and integrate a system (or a subsystem) and view its component-wise performance, and step-wise processing as well. The Dashboard is being successfully used by a consortium of eleven academic institutions to develop a suite of bi-directional machine translation (MT) systems for nine pairs of Indic languages, and six MT systems have already been deployed on web. The MT systems are being developed by reusing / re-engineering previously developed NLP modules, by different institutions, in different programming languages, using Dashboard as the testing and integration tool. The paper also discusses the experiences of developing MT products in consortium mode, using Dashboard as its integrating and testing platform, and its proposed enhancements.**

I. Introduction

As natural language processing (NLP) applications are knowledge intensive, complex, and are normally developed by a co-operating team of NLP experts, computational linguists, software engineers and language engineers, a large application like machine translation (MT) systems cannot be developed by a third party software vendor following classical software development paradigms [1]. For the development of NLP applications, the conventional software development tools are not very suitable as the application specifications are inherently imprecise, i.e., the output is not tested against 'correct output', but is validated against criteria specifying threshold of correctness, namely 'comprehensibility', 'understandability' or 'accuracy' [2]. In addition, an NLP application, unlike a conventional software application, goes through *continuous accuracy improvement, for considerable long duration, after their release*. Therefore, for the development of NLP applications, it is advisable to develop a Software Development Infrastructure [3] corresponding to the NLP software development paradigm being applied. The Dashboard Development Infrastructure, presented in this paper, is based on Blackboard architecture [4], an attractive option for building an NLP system, which facilitates integration of a set of heterogeneous modules (i.e., written in different programming languages) collaborating among themselves through a common in-memory data structure, referred as blackboard.

The need for creating a development infrastructure like Dashboard arose when the Technology Development for Indian Languages (TDIL) Group of Dept. of Information Technology (DIT[*]), Govt. of India formed a consortium of eleven academic & research institutions [5] for developing 18 pairs of ***ILMT Systems: Indian Languages to Indian Languages Machine Translation (ILMT) System*** these were over nine Indic languages, and the pairs (viz., Hindi↔{Bangla, Kannada, Marathi, Punjabi, Tamil, Telugu, Urdu}; Tamil↔Malayalam; and Tamil↔Telugu) were build by re-using / re-engineering most of the NLP components, and language resources that were available with the participating institutions. As eighteen ILMT systems were to be developed and deployed for public use in limited time frame of the project, the consortium considered various infrastructural approaches to facilitate the speedy implementation of these systems.

A. Infrastructural Approaches to Facilitate the Development of NLP Applications

To facilitate the development team comprising of NLP experts, computational linguists and software engineers, in building of large NLP applications, there have been three infrastructural approaches, viz., *frameworks, architectures,* and *development environments* [6]. The framework concept facilitates components based development, by providing a common and powerful platform with a number of mechanisms (e.g., ActiveX, Java Beans, etc.) that can be used, or adapted by the developers to build their systems. A framework greatly reduces development as well as maintenance effort, and it presumes that the system is going to be developed on common OS platform. An architecture defines a system in terms of its components, and the type of inter-relations among the components (e.g., client-server architecture, pipe-line architecture, etc.). It is the inter-relationships among the components that represent the distinguishing power of a specific architecture. Application developers have realized that a specific architecture is far more suitable for a family of software applications, giving rise to a concept of reference architecture or domain specific software architecture [7]. For NLP applications where heterogeneity of components is an

[*] We sincerely thank the TDIL (Technology Development in Indian Language) group, Dept. of Information Technology, Govt. of India in conceptualizing such a challenging project.

essential characteristic, blackboard architecture gets widely used [8]. Furthermore, when an implementation of an infrastructure based on a specific architecture provides additional tools for development / validation of a component, integrating and testing a set of components, and building and testing the complete system, it is usually called development environment [9].

B. Objective Requirement of a Development Environment for ILMT Systems

The consortium felt strongly for building a development environment as all participating institutions are geographically distributed, and multiple institutions have to collaborate and coordinate to deliver each ILMT system. As the chosen Indic languages have evolved from the classical language Sanskrit, the ILMT systems can be built on a common paradigm based *on Paninian framework* [10].

Further, the consortium decided to adopt the Blackboard Architecture [4] for building ILMT systems as it has to re-use previously developed set of heterogeneous modules. In black board architecture, heterogeneity of modules does not affect their operations as all of them operate on *common in-memory data structure*. Further, the consortium restricted the control to a *pre-specified lattice architecture*, as all the machine translation systems are being built following transfer based approach, in accordance with Paninian framework. The lattice is currently implemented as a pipeline compriseing of eleven major modules for each language pair.

All the major modules in the system consist of language independent *engines* and language specific parts. Among these eleven academic institutions, that are spread geographically, each institution was responsible for one or more specific engine, responsibility for each language rested with a single institution. Thus, language specific parts of all the modules for a specific language were the responsibility of a single institution. Therefore, the consortium needed a development tool that can be used by each institution independently, at different granularity levels of a system, i.e., either in re-engineering / testing of a module in isolation, or integration and testing of a subsystem, or building and testing of the complete system.

Correspondingly, the consortium visualized Dashboard, *a common blackboard based pipe-lined architectural framework* for building the translation systems which could utilize all available modules, in spite of their heterogeneity. And the *same framework could also be re-used,* as much as possible, so that all the eighteen ILMT systems have same architecture.

The Dashboard, apart from implementing the features of blackboard architecture, has many more additional features to enhance it as a development infrastructure for a family of NLP applications, such as Machine Translation System, Speech to Speech Translation System, Information Extraction System, etc. It is also equipped with a user-friendly visualization tool to build, test, validate, and integrate a system (or a subsystem), and view its step-wise processing and module-wise time profiling to facilitate the development team to improve system's accuracy and (speed) performance respectively.

C. The Dashboard Development Environment and the Present State of ILMT Systems

As all participating institutions were academic and research organizations, the consortium visualized a need for the participation of a software development company, called the Software Engineering Group (SEG), to supplement the tasks of professional software engineering practices to work with the consortium, *not as a vendor*, but in a symbiotic mode to facilitate them in engineering the ILMT systems as field deployable and maintainable products. The SEG, by symbiotically interacting with the participants, evolved the specification of the Dashboard development environment that would facilitate them in engineering / re-engineering available modules in symbiotic mode to build ILMT systems as stable products.

The first version of Dashboard development platform was released after one year, and since last two years, is being successfully used by the each participating institution independently. Out of the total eighteen MT systems being developed using Dashboard, the accuracy of four MT systems, viz., Punjabi→Hindi, Urdu→Hindi, Telugu→Tamil, and Hindi→Telugu, have improved considerably, and they have been deployed at website ***http://sampark.iiit.ac.in***. It is expected that remaining MT Systems would be released at regular intervals, spanning over next the six months.

Another consortium of seven academic institutions [11] which is developing Sanskrit to Hindi machine translation system is also using Dashboard as their system development environment.

The Technology Section of *Communication of the ACM (CACM)* [12] has compared these ILMT systems based on transfer based hybrid approach to that of purely statistical based approach of by Google and Microsoft. The Symbiotic Software Engineering / Re-engineering Paradigm facilitated by Dashboard, and followed in building the ILMT systems has also been reported in [13].

In Section 2, we discuss in detail the process the consortium followed to emphasize the need to develop the Dashboard Development Environment for building the suit of proposed ILMT systems. The requirement specification of the Dashboard Development Environment, and the way it is being used by members of consortium is given in Section 3. The Section 4 describes the cumulative experience of the consortium in using the Dashboard, and Section 5 gives the additional requirements that need to be incorporated in the future version of Dashboard to make it useful for any generic NLP application development.

II. THE NEED OF DASHBOARD DEVELOPMENT ENVIRONMENT FOR ILMT SYSTEM

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. The continuous involvement of NLP experts and computational linguists in the development process is essential as their knowledge cannot be passed to the software vendors. 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 the conventional software development tool, the development environment suitable for NLP applications has to cater, not only the specificity of NLP applications [14], but also the needs of each member type of the development team, viz., the NLP experts, the computational linguists and the software engineers.

A. Specificity of NLP Applications and the Requirement Specifications of Dashboard Development Environment

This subsection describes the salient requirements, emerging due to specificities of NLP applications [14] (distinct from conventional software), that must be covered by the proposed Dashboard development environment, and they are:

Heterogeneity of Modules: The complexity of NLP application is generally high as different component module works on different granularity 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 component module would need different data representation model for its efficient implementation. Also the different modules get developed in different programming language, the language that provides data representation model that suits its computation model making the implementation far natural, elegant and efficient. Sometimes for reasons of legacy software, different programming Hence, the development environment must facilitate development team to incorporate heterogeneous set of modules in the system.

Modularity: A set of generic NLP modules have been identified and developed and sometimes they are available off-the-shelf, from multiple sources. These generic modules are generally re-used for various applications and domains by proper adaptation and enhancements. This need requires that the development environment must view a system as built by integration of a set of independent modules.

Transparency at module interface level: The specifications for most NLP applications are approximate. Usually there is no concept of 'correct input' and a single 'correct output' [15]. For example, in an MT system grammatically wrong sentences are valid input as well as valid output. And hence, unlike conventional software, its output is not tested against 'correct output', but is validated against criteria specifying threshold of 'accuracy', mostly by human evaluators [2]. Therefore, a development environment should provide transparency at module interface level, (i.e., to view the input and the output of any module), facilitating the development team to easily isolate the module having the trouble spot, and independently modifying it, to improve the 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 code, 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.

Time Profiling of Modules: As the complexity of an NLP application is very high, performance is a major issue. Hence, time profiling of each component module is an imperative need which a development environment must provide so that the development team can concentrate to improve those modules that are time intensive.

Robustness of System: Some NLP components many a time, take quite a long time, get into indefinite loop, or even fail. Therefore, robustness at system level becomes another imperative need for any NLP application, and hence, the development 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.

B. The Blackboard Architecture

The project document [5] proposing to develop eighteen ILMT systems has stressed the need to develop all the systems based on pipe-lined blackboard architecture,

The blackboard in the classical form [4] can be viewed as a *central repository* of all shared information among multiple heterogeneous problem solving agents / *experts*. The name blackboard architecture was chosen to evoke a metaphor in which a group of experts gathers around a blackboard to collaboratively solve a complex problem. The information on blackboard represents assumptions, facts, events, and deductions made by system during course of problem solving. Each expert continuously sees the information on blackboard, and at some instant of time depending on the information content, if it feels it can contribute to the solution, tries to write (and update the information content) on the blackboard. At some stage, if multiple experts compete to write on blackboard, the *moderator or facilitator* controls the mediation among competing experts.

In case of the ILMT system, the *experts* are the *modules*, and the *central repository* is represented by *common in-memory data structure*. As the ILMT system is being built by following transfer approach following Paninian framework, currently the *moderator* limits the execution of modules in a pre-assigned order (i.e., pipe-line architecture) specified by the development team at the time of configuring the system.

C. Justifications of Blackboard Architecture for the ILMT Systems

The project document [5] has given objective and technical reasons that are enumerated below briefly for comprehensiveness of this paper:

Reuse of Natural Language Processing (NLP) Components and Language Resources: Since the project requires to reuse the NLP components and language resources (for most of the nine Indic languages) written in different programming languages (viz., Java, C, and Perl) and available with participating institutions, the blackboard infrastructure would be most suitable to build systems using independent heterogeneous modules.

Reuse of MT Architectural Framework: the consortium has *a common software architectural framework* of the MT system, viz., blackboard architecture, so that all the eighteen MT Systems have the same architecture. Again, this experience would be used in future to develop, MT systems covering other Indic language pairs on the same architectural framework.

Transfer Based Approach for ILMT Systems and Pipe-lined Blackboard Architecture: As all the ILMT systems are being built following transfer based approach, the consortium restricted the control to a *pre-defined pipe-line architecture*. Pipe-line architecture also reduces the complexity of configuring and testing systems under a development environment.

Graceful Degradation of System and Common In-memory Data Structure of the Pipe-lined Blackboard: In pipe-lined blackboard architecture modules communicate among themselves by its common in-memory data structure. This pipe-lined feature provides resiliency against failure of any in-between module of the system as following modules can still continue to work on the available in-memory data, assuring graceful degradation of the system.

Shakti Standard Format (SSF) for In-memory Data Structure of the Blackboard: Blackboard architecture is qualified by its in-memory data structure format. The consortium has adopted *Shakti Standard Format (SSF)* for in-memory data structure as it is based on tree structure with bags and associated feature structures [16], and has text notation for representing it unambiguously and providing human readability as well. Human readability of a module input/ output is an imperative need as it helps extensively in understanding the working and improving the module.

III. DASHBOARD DEVELOPMENT ENVIRONMENT AND ITS IMPLEMENTATION

Dashboard, as a framework for setting blackboards, was available with one of the participating academic institute [16, 17]. The SEG was assigned the task to redesign the Dashboard afresh, as a Development Environment with enhanced functionalities satisfying the ILMT systems' needs discussed above. The new Dashboard was required to be built on Linux platform, and the consortium concretized the requirement specifications of the first version of the proposed Dashboard Development Environment, which is given in next subsection.

A. Requirement Specifications of Dashboard Development Environment

1. The Basic Characteristics of Dashboard Development Environment
   a. The common in-memory data structure should be of Shakti Standard Format (SSF) [16],
   b. The system being built should be composed of a set of modules, where each module, independent of others, operates on the common in-memory data structure,
   c. The inter-module data exchange can be either through common in-memory data structure, or alternatively through an i/o stream,
   d. The set of modules should execute in a pre-specified sequence, i.e., following pipe-line architecture,
   e. It should provides application program interfaces (APIs), viz., reader / writer primitives on the common in-memory data structure (SSF) for programming languages C, Java, and Perl as most of the available modules are written in these languages.

2. Mechanisms to Configure a System to Run under Dashboard Development Environment
   a. Provision to define system specifications file to configure a system composed of set of modules in a pipe-line, granularity of pipe can be a single module, a set of module comprising a sub-system, or complete system,
   b. Provision to define modules' runtime properties through *module specification file*, viz., its programming language, i/o data format, i/o data encoding, i/o in stream or in-memory, level of module operation,
   c. Provision to edit *system specifications file* for reconfiguring the system, mainly to facilitate

c. Provision to edit *system specifications file*replacing a module with its new version,

d. Provision to compile a given system specification file, and generate the executable file of the *Moderator* which would be eventually executed to run the configured system under the Dashboard Development Environment.

3. Dashboard Development Environment as a Visualization Tool

   a. Display of intermediate i/o at module / subsystem level, and final i/o at system level, mainly to analyze the 'accuracy' of the output at module/ subsystem / system level,
   b. 'Step-by-step' run – interactively at module level,
   c. 'here and now' debugging tool – 'change the intermediate output' and run the system further. It would help to find multiple bugs in single execution, and it would also help in integration and system testing,
   d. Provision to save a session, i.e., to save the system level and each module level input/output, for post analysis by the development team,
   e. Provision for transliterating the input text (in source language script), or the output text (in target language script) of ILMT system, in either source language script or target language script, facilitating the single script readers to read both the input text as well as the output text.

4. Provision of get Time Profile for each sentence at module level.

5. Provision to specify Module Level Timeout for Failure Handling.*

6. Provisions of additional support tools for smooth inter-module interfacing, such as:

   a. Character encoding conversion tool, for converting data from UNICODE to WX and vice versa.
   b. Data format conversion tool, for those module input / output may be in different data format than that of the SSF. It provides tools to convert into, and convert from SSF to that acceptable to the module (e.g., SSF to TNT, etc.),
   c. i/o data validation tool, to verify whether the input / output data is in correct format or not.

As of today, Dashboard has been implemented with all the features envisaged above, (except the starred specification, i.e., the module level timeout). It is being used by all participating institutes independently, to integrate and test their modules, subsystem as well as the specific ILMT system(s) in which they are involved.

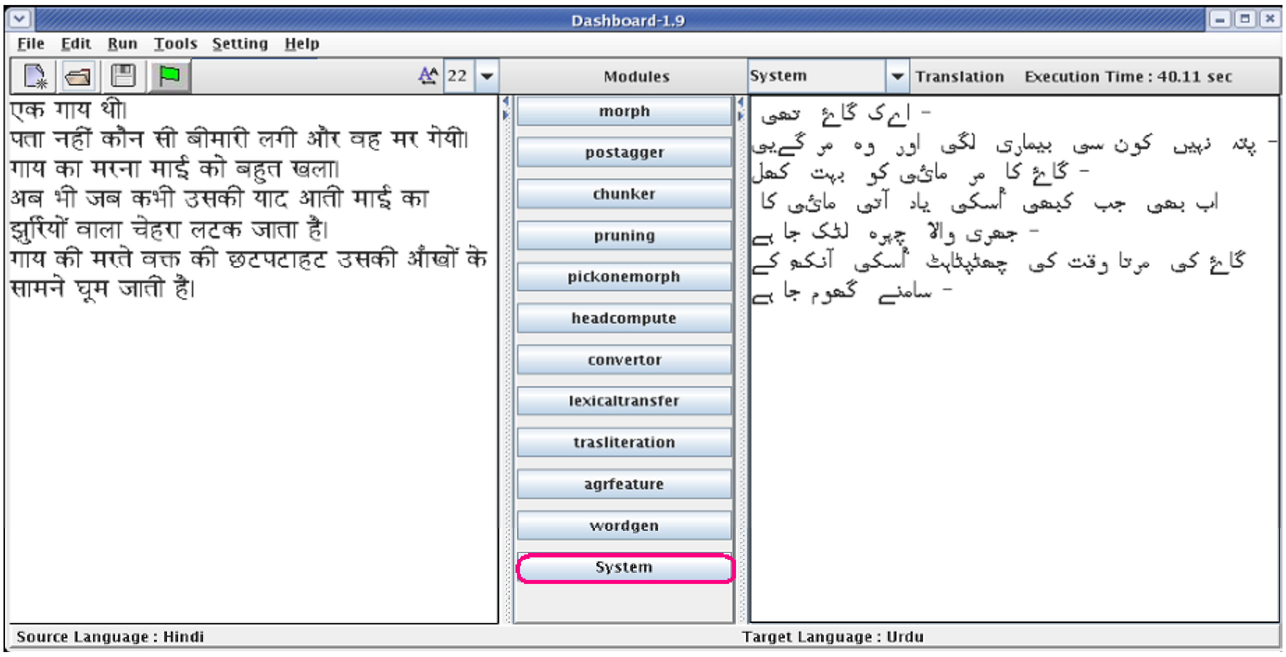
blackboard, build its own data structure, and after completing its task, it wishes back the new information in the common SSF blackboard.

B. The Working of Dashboard Development Environment: An Example

The power of the Dashboard Development Environment is illustrated through a set of screen shots given in Figures 1, 2, and 3. All these figures show the various screen shots of Hindi to Urdu ILMT system running under Dashboard. The vertical column in the middle provides the names of all eleven modules (and the last is always 'system') comprising the pipe-line architecture of the present system running under the Dashboard. The user of the Dashboard can provide input to the system either through a file, or typing directly on the left pane. Further, he can choose to run the complete system by clicking the 'system' choice from the vertical column. Alternatively, he can run the system 'step by step' by clicking the modules, one by one, from top to bottom.

Figure 1 shows the complete translation of Hindi text into Urdu as the user has chosen 'system' from the vertical column. The left pane shows the input text (composed of five sentences) written in the source language Hindi, and the right pane shows the translated output text, as the output of the ILMT system, written in the target language Urdu. The total system execution time of 40.11 seconds is also shown on the right corner of the Tool bar.

Once the system is executed completely, the session can be saved for future analysis. In case, the developer wants to analyze the intermediate output of any specific module, and for any specific sentence, he can do so through Dashboard. He has to choose first the sentence of his choice, by scrolling through the sentence numbers shown in the Tool bar adjacent

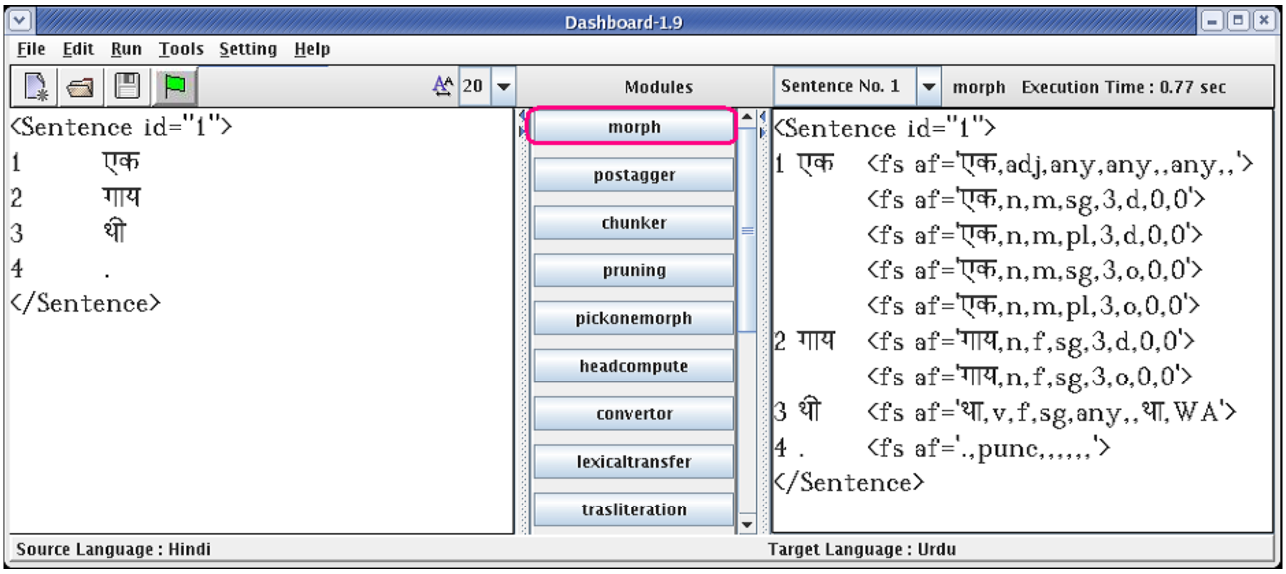


System | Translation Execution Time : 40.11 sec

एक गाय थी।
पता नहीं कौन सी बीमारी लगी और वह मर गेयी।
गाय का मरना माई को बहुत खला।
अब भी जब कभी उसकी याद आती माई का
झुर्रियों वाला चेहरा लटक जाता है।
गाय की मरते वक्त की छटपटाहट उसकी आँखों के
सामने घूम जाती है।

اےک گاۓ تھی -
پتہ نہیں کون سی بیماری لگی اور وہ مر گیےی -
گاۓ کا مر ماۓی کو بہت کھل -
اب بھی جب کبھی اُسکی یاد آتی ماۓی کا
جھری والا چہرہ لٹک جا ہے -
گاۓ کی مرتا وقت کی چھٹپٹاہٹ اُسکی آنکھو کے
سامنے گھوم جا ہے -

Figure 1: Dashboard showing Translation of Hindi text into Urdu



```
<Sentence id="1">
1       एक
2       गाय
3       थी
4       .
</Sentence>
```

Sentence No. 1 | morph Execution Time : 0.77 sec

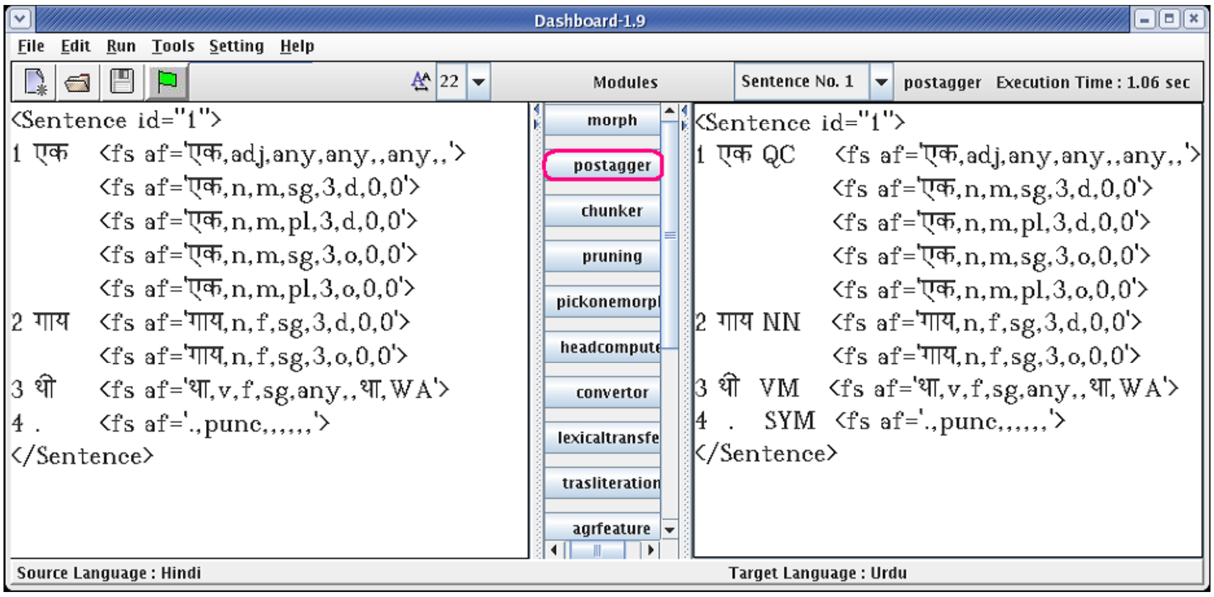
```
<Sentence id="1">
1 एक    <fs af='एक,adj,any,any,,any,,'>
        <fs af='एक,n,m,sg,3,d,0,0'>
        <fs af='एक,n,m,pl,3,d,0,0'>
        <fs af='एक,n,m,sg,3,o,0,0'>
        <fs af='एक,n,m,pl,3,o,0,0'>
2 गाय   <fs af='गाय,n,f,sg,3,d,0,0'>
        <fs af='गाय,n,f,sg,3,o,0,0'>
3 थी    <fs af='था,v,f,sg,any,,था,WA'>
4 .     <fs af='.,punc,,,,,,'>
</Sentence>
```

Figure 2: Dashboard showing Input and Output of Hindi Morph

to the execution time, and then click the module of his choice from the vertical pipeline. In Figure 2, the left pane shows the input (the first sentence in text form) to module 'morph' (the first module of the system pipe), and the right pane shows the output produced by the module 'morph' in SSF. The execution time consumed by the 'morph' module, i.e., 0.77 seconds, is also shown above.

Similarly, in Figure 3, the left pane shows the input (in SSF) to module 'postagger', and the right pane shows the output produced by it (in SSF). The postagger execution time of 1.06 seconds is also shown. In this way, the development team can post analyze the complete paragraph translation, by scrolling through each sentence level, and each module level executions.

For each specific inappropriate output, the *module level localization of fault* can easily be traced. Similarly, the module level time profile with sentence level granularity would easily guide the development team to *isolate those modules needing performance improvement.*



```
<Sentence id="1">
1 एक    <fs af='एक,adj,any,any,,any,,'>
        <fs af='एक,n,m,sg,3,d,0,0'>
        <fs af='एक,n,m,pl,3,d,0,0'>
        <fs af='एक,n,m,sg,3,o,0,0'>
        <fs af='एक,n,m,pl,3,o,0,0'>
2 गाय   <fs af='गाय,n,f,sg,3,d,0,0'>
        <fs af='गाय,n,f,sg,3,o,0,0'>
3 थी    <fs af='था,v,f,sg,any,,था,WA'>
4 .     <fs af='.,punc,,,,,'>
</Sentence>
```

```
<Sentence id="1">
1 एक QC    <fs af='एक,adj,any,any,,any,,'>
           <fs af='एक,n,m,sg,3,d,0,0'>
           <fs af='एक,n,m,pl,3,d,0,0'>
           <fs af='एक,n,m,sg,3,o,0,0'>
           <fs af='एक,n,m,pl,3,o,0,0'>
2 गाय NN   <fs af='गाय,n,f,sg,3,d,0,0'>
           <fs af='गाय,n,f,sg,3,o,0,0'>
3 थी  VM   <fs af='था,v,f,sg,any,,था,WA'>
4 .   SYM  <fs af='.,punc,,,,,'>
</Sentence>
```

Figure 3: Dashboard showing Input and Output of Hindi POS Tagger

C. Implementation Details of Dashboard Development Environment

The first version of Dashboard Development Environment was released in the latter part of 2008, and with continuous interactions with the participating institutions of the consortium, the SEG has released an enhanced version of Dashboard every six months. The latest version of Dashboard is fully stable, and it satisfies the major needs of consortium as the main development environment for productizing the proposed eighteen ILMT systems.

The back-end source code of the Dashboard is written in Perl, and the component handling the visualization interface is written in Java. The total size of complete Dashboard development Environment is 27.7 MB, out of which the program sizes of back-end component and its visualization interface are of 4.3 MB, and 23.4 MB respectively

IV. EXPERIENCE OF USAGE OF DASHBOARD DEVELOPMENT ENVIRONMENT

The consolidation of the experience of all eleven institutions (and the SEG) on their usage of Dashboard Development Environment to produce eighteen ILMT Systems through reusing / re-engineering their existing laboratory modules was done through a set of questionnaire distributed to them. This survey was done only after almost all eighteen ILMT systems were built by the consortium and were going through continuous accuracy and performance improvement by participating institutions. Four systems have been released on the web after their accuracy crossed the threshold of the acceptable limit. The major experiences are:

1. *As Engineering Tool* – A module written by an institution always runs satisfactorily in its own laboratory environment. Invariably, any other institution finds it extremely difficult to integrate the same module in their respective laboratory environment. This usually happens as context dependent hard coded data (e.g., path names of files, etc.) creeps into the code without researcher / developer being aware of it, making the module environment sensitive. Dashboard helps developer ***to engineer their module to make it portable***, as it does not permit any module to run under it with environment sensitive hard-coded data.
2. *As Testing and Integration Tool* – As the Dashboard was made available to each module developer, each one of them maintained their own testing environment by locally integrating their present version module with all other modules developed by other groups. In other words, it provided *a multiple replicated integrating and testing environment available with each of the participating institutes.* Whenever, a developer released next version of his module, all other developers upgrade their testing environment and test their systems with new released version of the module.
3. *As Group Testing and Coordination Tool* – The multiple replicated integrating and testing environments not only helped the group to test their module with real modules (and not with stubs), it also helped them ***to give feedbacks to the owners of preceding module*** (i.e., the module whose output they take as input) ***to improve the accuracy of the system***. Further, the Dashboard has become extremely helpful ***as a group coordination tool*** as well, because the members of the group are geographically scattered.
4. *As Profiling Tool* - ILMT system needs continuous improvement on its performance as well. The Dashboard provides time profile data of each module which helps ***to facilitate in improving the design of various time intensive modules***. In case, the response of the system falls short of user expectations, this *also gives guideline to*

enhance the h/w resources so that response to the user is brought within acceptable limit.

V. ENHANCEMENT VISUALIZED IN DASHBOARD DEVELOPMENT ENVIRONMENT

The feedback given by consortium, and interest shown by the development teams of other NLP applications, we have visualized a few essential enhancements to Dashboard to make it more useful for (i) generic NLP application development, (ii) large applications running on multiple machines, and (iii) system validation.

A. Dashboard Development Environment & Generic NLP Applications Development

Though the requirement specifications of Dashboard were derived from the specificity of generic NLP applications, the pressing needs of ILMT project constrained its specifications, making it useful more for the development of ILMT systems than a generic NLP application development environment. Presently, the Dashboard is tied with SSF, and its own way of writing system specification file. To make it generic NLP application development environment, it should have following features:

System specific In-memory Data Structure: It should provide system developers to define *system specific in-memory data structure* in other formates including XML.

System Specification: The *system specifications file* should be in XML where the modules dependency is specified for the system being setup.

B. Large Applications running on Multiple Machines

Partitioning: Dashboard can be distributed into multiple partitions (each partition comprises of a set of modules), and one Dashboard Partition can be configured to run on a specific machine.

Subsystem Specification: It should have provision to group the modules into subsystems.

C. System Validation

Validation Data Preparation: It will provide infrastructure to prepare and store validation data for each module as well as the system being setup.

Module Validation: It will have provision to perform module level validation.

Subsystem & System Validation: it will have provision to perform validation at subsystem as well as system level.

Regression Testing: It will provide mechanism to perform regression testing.

ACKNOWLEDGMENTS

We sincerely thank the TDIL (Technology Development in Indian Language) 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, CDAC Pune, IIIT Allahabad, Tamil University Tanjavure, IISc Bangalore) need special thanks who not only spelt out the requirements of Dashboard development environment, but used it continuously for their development, coordination, testing, building the ILMT systems, and also their continuous accuracy improvement. Their continuous feedback has not only improved its functionality, but also stabilized it as a product. The usability of Dashboard gets confirmed as the project has already delivered four systems with sufficient level of accuracy, and expects to deliver remaining fourteen ILMT Systems with sufficient level of accuracy within coming six months.

REFERENCES

[1] R.S. Pressman: Software Engineering: A Practitioner's Approach. 6th Edition, McGraw Hill International, 2006.

[2] R. Moona, S. Singh, R. Sangal, D.M. Sharma: MTeval: An Evaluation Methodology for Machine Translation Systems, Language Technology Research Center, IIIT Hyderabad, (2004).

[3] H. Cunningham, K. Humphreys, R. Gaizauskas, & Y. Wilks: Software Infrastructure for Natural Language Processing, Proceedings of 5th Conference on Applied Natural Language Processing, (1997).

[4] B. Hayes-Roth: A Blackboard Architecture for Control, Art. Intelligence, (1985).

[5] R. Sangal: Project Proposal to Develop Indian Language to Indian Language Machine Translation System, IIIT Hyderabad, TDIL Group, Dept. of IT, Govt. of India, (2006).

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

[7] L. Bass, P. Clements & R. Kazman: Software Architecture in Practice, 2nd Edition, Addison Wesley, 2003.

[8] D.D. Corkill: Blackboard Systems, AI Experts, Vol. 6, No. 9, pp: 40-47, Sept. 1991.

[9] H. Cunnigham, and J. Scot: Software Architecture for Language Engineering, Journal of Natural Language Engineering 10 (3/4), (2004).

[10] A. Bharati, V. Chaitanya, & R. Sangal: Natural Language Processing – A Paninian Perspective, Prentice Hall of India, New Delhi, (2004).

[11] Amba Kulkarni: Development of Sanskrit Tool Kit and Sanskrit to Hindi Machine Translation System (SHMT), Department of Sanskrit Studies, University of Hyderabad, Hyderabad, http://sanskrit.uohyd.ernet.in/shmt/login.php, accessed on May 28, 2010.

[12] G. Anthes: Automated Translation of Indian Languages, CACM, Vol. 53 (1), 2010.

[13] Pawan Kumar, Rashid Ahmad, Arun Kumar Rathaur, M.K. Sinha, R. Sangal: Reengineering Machine Translation Systems through Symbiotic Approach, Accepted in the 3rd International Conference in Contemporary Computing, NOIDA India, August 9-11, 2010. Proceedings to be published by Springer in Communications in Computer and Information Sciences, ISSN: 1865-0929.

[14] J.L. Leidner: Current Issues in Software Engineering for Natural Language Processing, HLT-NAACL 2003 Workshop on Software Engineering and Architecture of Language Technology Systems, (2003).

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

[16] R. Sangal: Architecture of Shakti Machine Translation System, IIIT Hyderabad, (2004).

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