INTERNATIONALIZATION IN GVIM

A PROJECT REPORT

Submitted by

Ms. Nisha Keshav Chaudhari
Ms. Monali Eknath Chim

In partial fulfillment for the award of the degree

Of

B. Tech Computer Engineering

UNDER THE GUIDANCE OF

Prof. Abhijit A. M.
College Of Engineering, Pune

DEPARTMENT OF COMPUTER ENGINEERING

AND INFORMATION TECHNOLOGY,

COLLEGE OF ENGINEERING, PUNE.

2009-2010

DEPARTMENT OF COMPUTERENGINEERING AND INFORMATION TECHNOLOGY, COLLEGE OF ENGINEERING PUNE

CERTIFICATE

Certified that this project titled **"INTERNATIONALIZATION IN GVIM"** has been successfully completed by "**Ms. NISHA KESHAV CHAUDHARI and Ms. MONALI EKNATH CHIM"** and is approved for the partial fulfillment of the requirements for the degree of "B. Tech. Computer Engineering".

SIGNATURE	**SIGNATURE**
PROF. ABHIJIT A. M.	**PROF. A. A. SAWANT**
Department of Computer Engineering,	Department of Computer Engineering,
College of Engineering	College of Engineering,
Shivaji Nagar,	Shivaji Nagar,
Pune- 411005	Pune- 411005

ACKNOWLEDGEMENT

We would like to thank our Project Guide, **Prof. Abhijit A. M.**, for his guidance, enthusiasm and inspiration during our complete project work. His helpful insights helped us to use several methods for tracking problems.

We would like to thank **Mr.Praveen Satpute** (Redhat), for giving valuable suggestions and guidance for our work.

Thanking All,

Nisha Chaudhari

Monali Chim

ABSTRACT

Today Information Technology is restricted to well English speaking people only. So if we want IT to reach masses then use of local languages in IT is essential. For using local languages Internationalization is must.

Internationalization is the process of designing a software for local languages so that people can use software in their own language. Internationalization is done by using Unicode.

GVim is an editor available on GNU/Linux and windows also. In GVim, there are some rendering issues of Devanagari script on Linux platform. Devanagari vowels do not get printed properly. We have fixed these problems to some extent.

TABLE OF CONTENTS

ACKNOWLEDGEMENT.......... ii

ABSTRACT.......... iii

1. LIST OF TABLES.......... 1

2. LIST OF FIGURES.......... 2

3. LIST OF ABBREVIATIONS AND NOMENCLATURE.......... 3

4. INTRODUCTION.......... 5

5. LITERATURE SURVEY.......... 6

5.1 Unicode.......... 6

5.2 UTF-8.......... 7

5.3 UTF-16.......... 7

5.4 Pango.......... 9

6. SPECIFICATION.......... 10

6.1 Problem definition.......... 10

6.2 Software Requirement.......... 10

7. IMPLEMENTATION.......... 11

8. TESTING METHODOLOGIES.......... 16

9. CONCLUSION.......... 18

10. FUTURE ENHANCEMENT.. 19

REFERENCES.. 20

1. LIST OF TABLES

1.1 The Unicode Standard, Version 5.2, Archived Code Charts………..…………......8

2. LIST OF FIGURES

1.1 GVim before making changes……………………..……………………………….....10

1.2 GVim after making changes….………………..…………………………………15

1.3 GVim after testing……………………..……..………………………………...17

3. LIST OF ABBREVIATIONS AND NOMENCLATURE

GVim

GVim is a GUI version of vim. Vim is an enhanced version of the Unix vi editor.

GUI

A user interface based on graphics (icons and pictures and menus) instead of text.

GTK (Gimp Tool Kit or GUI Tool Kit+)

A library of object-oriented graphical interface elements for developing X Window applications in C/C++, Python, Perl and other languages.

Vim

Vim is a highly configurable text editor built to enable efficient text editing. It is an improved version of the Vi editor distributed with most UNIX systems. We can get the description of the Vim at www.vim.org.
Also, there is community for Vim developers at vim-dev@vim.org.

GDK

GDK (GIMP Drawing Kit) is a computer graphics library that acts as a tool for the low-level drawing and windowing functions provided by the underlying graphics system.

Pango

Pango is open source software that seeks to create a software framework so that international text characters can be electronically rendered.

Unicode

The Unicode Standard is the universal character encoding standard used for representation of text for computer processing. Unicode provides a consistent way of encoding multilingual plain text making it easier to exchange text files internationally.

UTF-8

UTF-8 stands for **U**nicode **T**ransformation **F**ormat-**8**.

UTF-8 encodes each Unicode character as a variable number of 1 to 4 octets, where the number of octets depends on the integer value assigned to the Unicode character.

UTF-16

The encoding form maps each character to a sequence of 16-bit words. Characters are known as code points and the 16-bit words are known as code units.

Glyph

In information technology, a glyph (pronounced GLIHF; from a Greek word meaning carving) is a graphic symbol that provides the appearance or form for a character. A glyph can be an alphabetic or numeric font or some other symbol that pictures an encoded character

4. INTRODUCTION

Today there are many applications that provide the feature of editing in local language. Many of the applications successfully provide this feature but some applications like GVim provide this feature partially that is they edit in local language but with some problems of rendering, etc. In windows this problem of using local language has been tried to be solved by using different softwares like Akriti, Shreelipi etc but these softwares have created a new problem of standardization as Unicode is not used in these softwares.

On Linux platform, this problem is solved by using Unicode support .Unicode provides unique number for each character in the world. By using Unicode, Internationalization work is done.

Internationalization term is frequently abbreviated to the i18n (where 18 stands for the number of letters between the first *i* and last *n* in *internationalization).*
I18N is needed in the following places.

1. Displaying characters for the users' native languages.
2. Inputting characters for the users' native languages.
3. Handling files written in popular encodings that are used for the users native languages.
4. Using characters from the users' native languages for file names and other items.
5. Printing out characters from the users' native languages.
6. Displaying messages by the program in the users' native languages.

5. LITERATURE SURVEY

For modifying the code we need to study the following terms.

5.1 Unicode

Fundamentally, computers just deal with numbers. They store letters and other characters by assigning a number for each one. Before Unicode was invented, there were hundreds of different encoding systems for assigning these numbers.

These encoding systems also conflict with one another. That is, two encodings can use the same number for two different characters, or use different numbers for the same character. Any given computer (especially servers as they need to store data in different encodings) needs to support many different encodings, yet whenever data is passed between different encodings or platforms, that data always runs the risk of corruption.

No single encoding could contain enough characters. Even for a single language like English, no single encoding was adequate for all the letters, punctuations, and technical symbols in common use.

Unicode provides a unique number for every character, no matter what the platform, no matter what the program, no matter what the language. It is supported in many operating systems, all modern browsers, and many other products.

Unicode enables a single software product or a single website to be used on multiple platforms, languages and countries. It allows data to be transported through many different systems without any changes.

5.2 UTF-8

UTF-8 stands for Unicode Transformation Format-8. It is an octet (8-bit) encoding of Unicode characters. UTF-8 encodes each Unicode character as a variable number of 1 to 4 octets, where the number of octets depends on the integer value assigned to the Unicode character. It is an efficient encoding of Unicode documents that use mostly US-ASCII characters because it represents each character in the range U+0000 through U+007F as a single octet. UTF-8 is the default encoding for XML.

5.3 UTF-16:

In computing, UTF-16 (16-bit UCS/Unicode Transformation_Format) is a variable-length character encoding for Unicode, capable of encoding the entire character set. The encoding form maps each character to a sequence of 16-bit words. Characters are known as code points and the 16-bit words are known as code units. All possible code points from U+0000 through U+10FFFF, except for the code points U+D800–U+DFFF (which are not characters), are uniquely mapped by UTF-16 regardless of the code point's current or future character assignment or use.

Table of Unicode Standard Chart for Devanagari is as shown below in Table 1.1:

	090	091	092	093	094	095	096	097
0	◌ऀ 0900	ऐ 0910	ठ 0920	र 0930	◌ी 0940	ॐ 0950	ॠ 0960	॰ 0970
1	◌ँ 0901	ऑ 0911	ड 0921	ऱ 0931	◌ु 0941	◌॑ 0951	ॡ 0961	ॱ 0971
2	◌ं 0902	ऒ 0912	ढ 0922	ल 0932	◌ू 0942	◌॒ 0952	◌ॢ 0962	ॲ 0972
3	◌ः 0903	ओ 0913	ण 0923	ळ 0933	◌ृ 0943	◌॓ 0953	◌ॣ 0963	▨
4	ऄ 0904	औ 0914	त 0924	ऴ 0934	◌ॄ 0944	◌॔ 0954	। 0964	▨
5	अ 0905	क 0915	थ 0925	व 0935	◌ॅ 0945	◌ॕ 0955	॥ 0965	▨
6	आ 0906	ख 0916	द 0926	श 0936	◌ॆ 0946	▨	० 0966	▨
7	इ 0907	ग 0917	ध 0927	ष 0937	◌े 0947	▨	१ 0967	▨
8	ई 0908	घ 0918	न 0928	स 0938	◌ै 0948	क़ 0958	२ 0968	▨
9	उ 0909	ङ 0919	ऩ 0929	ह 0939	◌ॉ 0949	ख़ 0959	३ 0969	ॹ 0979
A	ऊ 090A	च 091A	प 092A	▨	◌ॊ 094A	ग़ 095A	४ 096A	ॺ 097A
B	ऋ 090B	छ 091B	फ 092B	▨	◌ो 094B	ज़ 095B	५ 096B	ॻ 097B
C	ऌ 090C	ज 091C	ब 092C	◌़ 093C	◌ौ 094C	ड़ 095C	६ 096C	ॼ 097C
D	ऍ 090D	झ 091D	भ 092D	ऽ 093D	◌् 094D	ढ़ 095D	७ 096D	ॽ 097D
E	ऎ 090E	ञ 091E	म 092E	◌ऻ 093E	◌ॎ 094E	फ़ 095E	८ 096E	ॾ 097E
F	ए 090F	ट 091F	य 092F	◌ि 093F	▨	य़ 095F	९ 096F	ॿ 097F

5.4 Pango

Pango is a library for laying out and rendering of text, with an emphasis on internationalization. Pango can be used anywhere when text layout is needed, though most of the work on Pango so far has been done in the context of the GTK+ widget toolkit. Pango forms the core of text and font handling for GTK+-2.x.

Pango is open source computing library used by software developers for laying out and rendering text in high quality, emphasizing support for multilingual text. Different font back-ends can be used, allowing cross-platform support; so that Pango-rendered text will appear similar under different operating systems, such as Linux, Apple's MacOS and Microsoft Windows.

Pango has been integrated into most Linux distributions. It also provides the rendering for text in the Mozilla Firefox web browser.

The Pango project provides a highly modular framework for internationalized text layout and rendering, with the ability to incrementally add support for both new scripts for new font technologies and rendering systems.

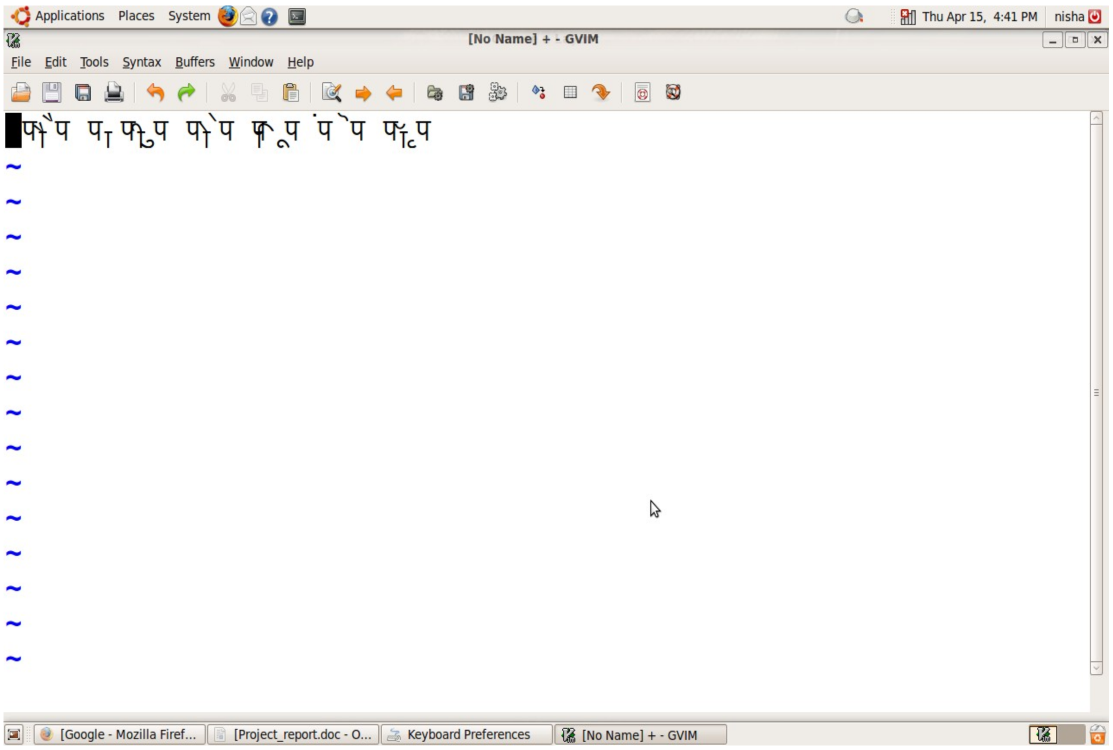
We can also get detailed information about the pango at www.pango.org.

6. SPECIFICATION

6.1 Problem definition

To solve rendering problem of Devanagari script in GVim Editor on Linux platform.

These rendering problems can be seen in the following snapshot of Gvim. As we can see Devanagari characters do not get printed properly especially vowels.



Applications Places System
Thu Apr 15, 4:41 PM
nisha
[No Name] + - GVIM
File Edit Tools Syntax Buffers Window Help
[Google - Mozilla Firef...
[Project_report.doc - O...
Keyboard Preferences
[No Name] + - GVIM

Figure 1.1

6.2 Software Requirement

Ubuntu 9.04

GVim Source Code

Pango Library

GTK Library

7. IMPLEMENTATION

How to install GVim :

To implement the changes in GVim related to rendering of the Devanagari script we need to go through following steps.

There are two types of packages for installing software in Ubuntu:

1. Debian package (.deb)
2. Source package (.tar.gz etc..)

No source files are there in Debian package.

We can install the software by configuring it using source package. These source packages are in compressed form having extension like .tar.gz. We have to extract this source package by using command

```
tar -xvzf <package name>
```

Then it will create a new folder containing extracted files. To install the GVim we need to follow the steps given below:

Get the source code from the mercurial repository:

https://vim.googlecode.com/hg/

This repository is recommended because it contains all source files needed to install GVim.

Then extract files as stated above.

Run the following commands to install GVim application:

1. To compile the GVim code
   ```
   cd vim
   cd src
   configure
   make
   ```
2. To install the VIM binaries in your path:
   ```
   sudo make install
   ```

To run the GVim application use the command

gvim or *vim -g <filename>*

Environment Settings:

Keyboard Layout:

For writing in Devanagari script, we need to set keyboard layout to Devanagari script.

This is done as follows:

1. Go to System->Preferences->Keyboard
2. Select keyboard layout .Go to Add option.
3. Then select country -> India and then select language.
4. The Variant should be India or India Hindi Bolnagri.

GVim source code modifications:

GVim contains different files and folders and makefiles. We can make the changes in GVim application by modifying files in src folder. This src folder contains different c files, header files and makefile. The c files are of different types. Some files are specific for GUI application of Vim. And some are common for GVim and Vim. GUI specific file starts with "gui" in their name.

In GUI files functions of pango are used to display the characters on the screen. By making changes in the functions of the GUI files we can change the display of the characters. So changes are done in GUI files.

File named `gui.c` contains the structure of all the GUI information. `gui_gtk_x11.c` file contains various functions that are used to display characters on the screen. Many pango functions are used here. It helps to display characters from different scripts. GVim gets the different script characters in Unicode format and then it sends those Unicode to pango then pango retrieves the glyph for that character. The glyph is then passed to GVim and then it adjusts the glyph by using rendering engine pango and then displays on the screen.

Following changes are made:

1. Changes are made in the function called `setup_zero_width_cluster` in `gui_gtk_x11.c` file. `x_offset` of glyph is set to zero here, previously it was -8 since `glyph->geometry.x_offset = - width + MAX(0, width - last_cluster_width) / 2;`

Here `width` is 8 and `(MAX(0, width - last_cluster_width) / 2)` is 0. Since `x_offset` of glyph was -8 previously it prints the composing character previous to the character where it should actually print.

If we change `x_offset` of the glyph to +8, it prints composing character next to the character where it should actually print.

2. In the same function one pango function is called named `pango_font_get_glyph_extents`. This function gets the logical and ink extents of a glyph within a font. The coordinate system for each rectangle has its origin at the base line and horizontal origin of the character with increasing coordinates extending to the right and down.

The units of the rectangles of the glyph are in `1/PANGO_SCALE` of a device unit. This function aligns the characters to the origin of the rectangle of each character. Because of this composing characters are getting printed slightly down where it should actually get printed. We don't need this for Devanagari script. It may be for some other scripts. So if we don't call this function we can fix this problem. It prints the Devanagari characters properly.

3. In the same file `gui_gtk_x11.c`, in the function *`gui_gtk2_draw_string`*, there was a same problem (as mentioned above in the first point) of `x_offset` of the glyph. So we have increased `x_offset` of the glyph so that it will print properly. It has been given the value of width of character.

4. In function `draw_glyph_string`, function is called to define the rectangle and then to draw the glyph in that rectangle by calling function `gdk_draw_glyphs`. In this function value of `TEXT_X(col)` is passed. `TEXT_X` converts character column into X pixel coordinate for drawing strings. We have shifted the glyph in defined rectangle to the left by 2.

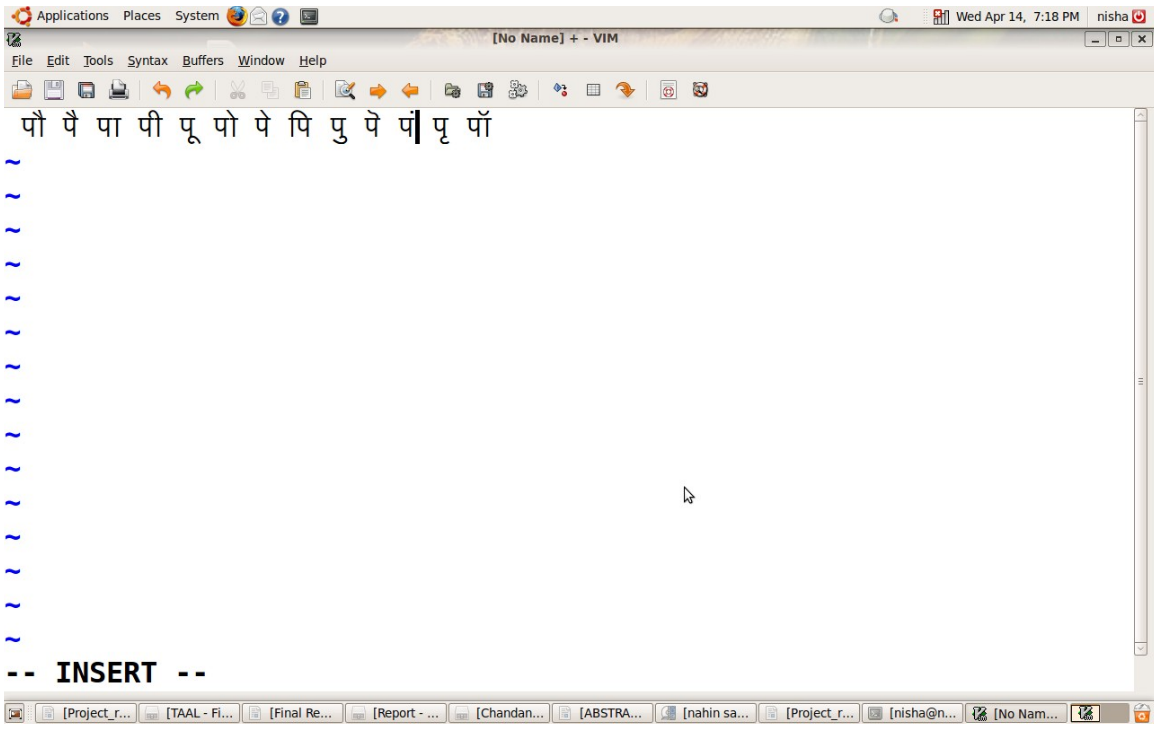
If we don't shift to the left, then composing character enters the rectangle of the next character and the next character doesn't get printed properly. We solved this problem by shifting glyph to the left.

5. ि was getting printed to the next character. So to fix this problem we have checked that the character is ि or not by using the Unicode 0x93F. Then according to that we changed the `x_offset` of this character in the `gui_gtk_x11.c` file.

6. When we made all these changes, other scripts were getting affected by this modified code. So we checked the input characters if they are Devanagari characters or not. We have checked whether the input characters are in the range 0x900 – 0x97F. Hence the changes will not affect rendering of other languages like Arabic, Tamil, Telugu, English, etc.

Diff of the old code and the new source code of GVim is published at gvim-dev@blogspot.com.

After making changes we can see that the Devanagari script gets printed properly as shown in the below snapshot of the GVim:



```
पौ पै पा पी पू पो पे पि पु पे पं पृ पॅ
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

Figure 1.2

8. TESTING METHODOLOGIES AND RESULTS

Testing performed:

In simple words, testing is the process of verification of the code to check whether we are getting the desired output for each and every valid input.

We have done changes in GVim source code and these changes need to be tested.

When we input ो after some consonant, it was printing ो slightly down. We have modified the code and now it prints correctly. We have shifted glyph of each Devanagari character to the left to display Devanagari characters properly. We changed the offset for displaying character 0x93F (ि). We have done these changes only for Devanagari script.

After modifying the code, we compiled the code and ran the application.

Following are the test cases:

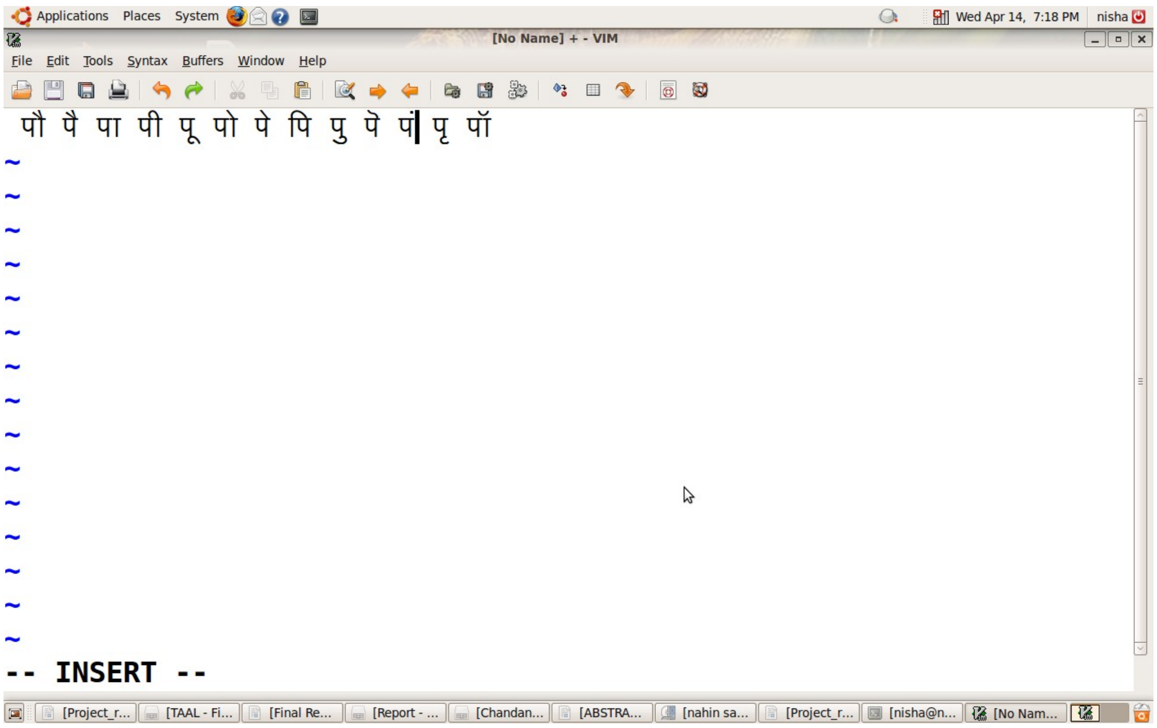
1. If we first type प, then ि then it should output पि not प ि. For every character in Devanagari, it should output ि to the desired character not the next character.

2. Then, if we type English characters, there should be no change in the display of those characters.

3. If we type प and ो then it should print पो. Same is the case for all vowels of the Devanagari script.

4. Also when we input vowel, the next character to it should get printed completely. It should not overwrite the next character or next character should not get printed partially.

Screen shot of the tested GVim code is as follows:


Applications Places System
Wed Apr 14, 7:18 PM nisha
[No Name] + - VIM
File Edit Tools Syntax Buffers Window Help
[Project_r... [TAAL - Fi... [Final Re... [Report - ... [Chandan... [ABSTRA... [nahin sa... [Project_r... [nisha@n... [No Nam...

```
 पौ  पै  पा  पी  पू  पो  पे  पि  पु  पे  पं  पृ  पॉ
~
~
~
~
~
~
~
~
~
~
~
~
~
~
~
-- INSERT --
```

Figure 1.3

9. CONCLUSION

We have solved some rendering problems of Devanagari script in Gvim on Linux platform. This will be very useful to those people who want to use Devanagari script for editing in GVim. It will help IT to reach common people.

10. FUTURE ENHANCEMENT

1. We can make changes in the code so that we can print the complicated combining characters using ◌्.
2. We can improve the display of characters *ksha, pra, dnya.*
3. The character ◌ु should get printed at proper place when we print it after र.
4. Also we can create more spacing between the characters for better display.

REFERENCES

1. http://en.wikipedia.org/

2. http://www.pango.org/

3. http://www.vim.org/

4. Mercurial Repository https://vim.googlecode.com/hg/

5. The Unicode Standard, Version 5.2, Archived Code Charts

6. vim-dev@vim.org
   Community for the Vim Development