Об интерфейсе

Д. Анисимов

24 июля 2004 г.

Содержание

1 Почему сделано так как сделано.

Способ существования и разработки Open-Source проекта имеет существенные отличия от способа существования и разработки коммерческого проекта. Разработчики работают в свободное время, с разными скоростями, могут "внезапно выйти из проекта". Война, пожар, любовь, сессия - все эти причины приводят к тому, что состав команды очень переменный - люди приходят, уходят, возвращаются... На это можно отреагировать разными способами.

В проекте Меркурий-Правда проблемма "текучки"решается тем, что есть очень небольшое ядро "постоянных участников", которые и несут ответственность за поддержание кода временно выбывших помощников. По-этому архитектура и стиль существенно заточены под привычки и комплексы "постоянных участников".

Когда мы решали "как делать", мы руководствовались следующими соображениями:

1. Программа (и ее интерфейс) должна быть написана простыми средствами. Вновь прибывший человек должен тратить минимум времени для изучения используемых средств. После убывшего человека не должно оставаться "непонятного кода".
2. Программа (и ее интерфейс) должна быть легко переносима. Мы хотим иметь один комплект исходников и для X-версии, и для терминала, и для OS/2 - Windows.
3. Интерфейс должен быть достаточно надежным и "нетормозным". Чтобы разработчики ядра перевода "отлавливали только свои ошибки".
4. Ну и интерфейс должен быть достаточно удобным для разработчиков и пользователей.

Рассматривались три альтернативы: Qt, gtk+ и доморощенная video.cpp. Против Qt и gtk+ было то, что они имеют "более, чем 20 функций", и по-этому трудно передавать код от человека к человеку. Qt кроме того меняет набор функций "чаще чем раз в 5 лет". По сумме показателей на ближайшее время волевым решением была выбрана доморощенная video.cpp[1]. Этот выбор предопределил внешний вид программы - "подражание Нортону".

2 Библиотека video.cpp

Чтобы начать работать с video.cpp достаточно уметь выполнять всего 4 функции.

1. Начинать работу,
2. Заканчивать работу
3. Написать что-то в заданном месте экрана
4. Принять код нажатой клавиши.

2.1 Необходимые функции.

void s_begin_schone(void);
void s_begin_schone(int argc, char **argv);

Начать работу.

void s_end_schone(void);

Завершить работу.

void s_get_size(short &sx, short &sy);
short s_get_sx(void);
short s_get_sy(void);

Узнать размер окна программы.

void s_text_yxf(short y, short x, uchar f, char *str);

Напечатать текст в определенной позиции.

void s_text_yxf1(short y, short x, uchar f, char *str);

Напечатать текст в определенной позиции в кавычках.

void s_text_yxfl(short y, short x, uchar f, short L, char *str);

Напечатать текст в определенной позиции (но строка не обязана заканчиваться нулем). Очень удобная функция, когда хочется выделить какую-то часть строки (слова)

void s_text_yx(short y, short x, char *str);

Напечатать текст но не изменять цвет в этой части экрана.
y x - координаты начала выводимой строки. Отсчет начинается с нуля. Отсчет y - с верха.
f - фон и цвет выводимого текста. Первые 4 бита - цвет фона, последние 4 бита - цвет букв. Цвета имеют следующие значения:

[1]на ближайшее время - это значит пока "Правда"не начнет прилично переводить. А когда начнет, интерфейс будет не сильно важен. Народ вообще просит "Правду"в виде библиотеки.

0x00 - черный
0x01 - красный
0x02 - зеленый
0x03 - желтый
0x04 - синий
0x05 - фиолетовый
0x06 - голубой (морская волна)
0x07 - (темно) белый
0x08 - ярко черный (это светлее, чем темно белый)
0x09 - ярко красный
0x0a - ярко зеленый
0x0b - ярко желтый
0x0c - ярко синий
0x0d - ярко фиолетовый
0x0e - ярко голубой
0x0f - ярко белый

L - длина строки

void s_goto_xy(short y,short x);

Установить курсор в позицию y, x.

void s_getch(short *,short *);

Дождаться и принять нажатую клавишу. Функция несколько старомодная, по-этому читайте внимательно. Использовать эту функцию надо так:

```
short key1,key2 ;
s_getch( &key1, &key2 );
```

Если нажата алфавитно-цифровая клавиша, то код "буквы"будет помещен в key1. Если нажата "служебная"клавиша (стрелки, F1-F10, Home и т.д.), то в key1 записывается 0, а код клавиши, записывается в key2. Коды спецклавиш можно посмотреть в файле s_defkey.h

Почему так сделано? Потому что так было сделано в ДОС-е. А поскольку живу я долго, и эта метода не хуже, чем любая другая, то пущай так и остается...

int s_shiftstatus(void);

Определить какие модификаторы (Shift, Ctrl, Alt) были нажаты при нажатии очередной клавиши. Для букв и цифр это не важно, а вот всякие Ctrl+Home таким образом определять удобно. Пользоваться этой функцией надо примерно так:

```
short SS = s_shiftstatus( );

if( SS & (S_Alt_L|S_Alt_L ) ) // если нажат левый или правый Альт
   ; // действие 1
if( SS & (S_Ctrl_L|S_Ctrl_L ) ) // если нажат левый или правый Контрол
   ; // действие 2
if( SS & (S_Shift_L|S_Shift_L ) ) // если нажат левый или правый Шифт
   ; // действие 3
```

2.2 Другие полезные функции.

void s_nacht(void);

Сделать весь экран черным.

void s_rame1_f(short y1, short x1, short y2, short x2, uchar f);
void s_rame1_F(short y1, short x1, short y2, short x2, uchar f);
void s_rame2_f(short y1, short x1, short y2, short x2, uchar f);
void s_rame2_F(short y1, short x1, short y2, short x2, uchar f);

Нарисовать рамку. y1, x1, y2, x2 - координаты. f - цвет. Функции с циферкой 1 - рисуют одинарную рамку, с циферкой 2 - двойную. Функции с буковкой F - затирают внутреннюю область рамки, с буковкой f - не затирают.

void s_save (short y1, short x1, short y2, short x2);

Запомнить (внутри библиотеки) прямоугольную область экрана. Функция слегка опасная, ибо буфер может переполниться, а библиотека этого не контролирует.

void s_save_rame(short y1, short x1, short y2, short x2);

Запомнить (внутри библиотеки) "то, что под рамкой".

void s_restore(void);

Восстановить "то что было запомнено"предыдущей функцией s_save или s_save_rame. На один вызов s_save или s_save_rame должен приходиться один вызов s_restore (как скобки). Обычно я это делаю при входе и выходе из функции.

2.3 Функции "только для начальства".

void s_set_size(short sx, short sy);

Установить размер окна программы (в буквах). Работает только в X-версии. В остальных версиях игнорируется.

void s_set_font(char *Name);

Установить шрифт.

char *s_get_font(void);

Спросить какой фонт сейчас установлен.

void s_refresh(void);

Перерисовать картинку на экране. Почему-то корректно работает только в терминальной версии.

void s_set_ruskomb(short Komb);

Установить какой комбинацией переключаются русские буквы.

2.4 Экзотические и бесполезные функции.

void s_color_yxt(short y, short x, uchar *t, char *str);
void s_foreground_yxt(short y, short x, uchar *t, char *str);

Довольно экзотичные функции - установить цвет и установить фон в строке начиная с позиции y x. Цвет и фон задается отдельно для каждой буквы. t - массив цветов (значения цветов описаны выше) str - массив индексов цветов. Индексы начинаются с нуля и заканчиваются "f". Если написать вот так:

```
uchar color[]={ 1, 6, 15 };
s\_color\_yxt( y, x, color, "00011211022" );
```

то строка, находящаяся в y, x будет переливаться всеми цветами радуги :-)

void s_redraw(void);
void s_redraw_str(short y);
short s_clear_cursor(void);
short s_set_cursor(short S);

Сам забыл что это такое. значит настолько нужно :-)

3 Методология "цикл обработки сообщений".

В этом стиле интерфейс было модно писать во времена ДОСа. При этом морда получается несколько "простоватая", но зато эта методолгия позволяет писать с абсолютно минимальными усилиями. Для нас это актуально, поскльку основные силы уходят на алгоритм перевода. Вот как выглядит типичная интерфейсная функция:

```
e_WinMsg t_InterFaceClass :: main_loop( )
{
   init_function(); // присвоение начальных данных

   while( 1 )      // основной цикл работы
   {
      paint();      // рисование текущего состояния программы

      s_getch( &key1, key2 ); // прием нажатой клавиши
      SS=s_shiftstatus( );    // и ее клавиш модификаторов

      switch( key1 )
      {
         case '1' :
            ...      // обработка нажатий клавиш
            break ;
```

```
 case 0 :
    switch( key2 )
    {  case S_key_Up :
            ...     // обработка нажатий служебных клавиш
            break ;
    }
    break ;
      }
   }
   final_function();    // результаты работы диалога
   return Ret ;
}
```

main_loop() - это функция, которая реализует какой-нибудь режим работы программы (например выбор варианта перевода фразы).

init_function() - функция построения данных для этого режима (например дерева вариантов перевода фразы).

paint() - рисование текущего состояния программы. Например, текущего выбранного варианта переводимой фразы, и множества возможных продолжений фразы. Рисование не обязательно делать именно одной функцией.

s_getch() и s_shiftstatus() наверное не требует особых пояснений.

Оператор switch(key1) обеспечивает изменение внутренних переменных класса t_InterFaceClass под воздействием нажатых кнопок. Собственно, можно выполнять любые действия над данными программы, но "правилом хорошего тона"является не выходить за пределы переменных класса. В этот switch() никогда не приходят сигналы перерисовки. За перерисовку отвечает библиотека video.cpp. В этот switch() никогда не приходят сигналы смены режима (смены текущего окна). Режимы переключаются только по инициативе самого текущего режима.

final_function() финальные действия. Например занесение выбранного варианта фразы во внутреннюю строку класса. Полученными значениями переменных воспользуется вызывающий режим.

Функция возвращает одно из следующих значений:

WM_NEXT перейти в следующее окно
WM_PREV перейти в предыдущее окно
WM_ESC выйти из режима (неуспешно)
WM_OK выйти из режима успешно
WM_HELP перейти в режим "подсказка"

Есть и другие возвращаемые значения, но новичкам их лучше не использовать.

В программе "Правда"весь интерфейс написан по этой методологии.

То есть где-то есть главный цикл обработки нажатых кнопок. Некоторые нажатия приводят к переходу в другие режимы. В другом режиме тоже есть своя функция отрисовки, и функция обработки кнопок. Этот другой режим может вызвать какой-то третий режим, или вернуться в предыдущий. Таким образом всегда есть "стек режимов", из которого всегда можно перейти в следующий режим или вернуться в главный режим.

4 Описание классов интерфейса.

Все интерфейсные классы можно разбить на три группы

Классы, которые используются юзером при переводе

t_Text этот класс невидим пользователю. Этот класс хранит внутри себя текст оригинала или текст перевода, и отвечает за все операции над текстом (как массивом букв).

t_Edit текстовый редактор. Этот класс отвечает за интерфейс с пользователем, во время редактирования текста. Он же используется для отображения подсказки (Хелпа). Внутри себя этот класс содержит t_Text.

t_EditTrans текстовый редактор, заточенный на перевод. Является производным от t_Edit. Отличается тем, что имеет несколько добавочных действий. (например перейти к следующей фразе)

t_Menu Класс меню.

t_Path Диалог выбора варианта перевода фразы.

t_Option класс, отвечающий за считывание настроечного файла, и хранение настроечных переменных.

t_Dir диалог открытия и записи файла.

t_HelpData класс хранящий подсказки. (Именно хранящий, за отображение отвечает t_Edit)

t_AddWord диалог добавления нового слова в словарь.

t_Windows главный интерфейсный класс, который отвечает за управление всеми другими окнами. (Никакого отношения к МС-Виндовс)

Отладочные классы, которые используются при отладке

t_Debug отображение главной таблицы перевода (окно "отладчик")

t_GrammarView окно "база грамматики"

t_Slowo3View отображение содержимого словаря окно "база переводов"

t_Edit2 отображение первичного массива слов (окно "первое чтение")

t_EditPart окно "выбор части речи"когда нажмешь Enter в "первом чтении"

t_Edit3 отображение структуры фразы (окна "структура источника"и "структура приемника")

t_SlowoView отображение массива словарей (окно "база словарей")

t_SlowoView1 отображение грамматических форм, присутствующих в словаре (когда нажмешь Enter в "базе словарей")

t_EditForm редактор грамматических форм (сам забыл, по моему не используется)

t_EditForm1 редактор грамматических форм (вспомогательный)

Служебные классы, которые находятся глубоко в потрохах

t_Win абстрактный класс окна. Хранит в себе общие данные нужные для любого окна программы - размеры окна и название.

t_SelectWin абстрактный класс списка

5 Что же делать тем, кто знает GTK+ или другие библиотеки?

Возможно, когда-нибудь мы перепишем интерфейс на что-нибудь более красивое и удобное, чем "голая консоль в стиле 80-х годов прошлого века". Но произойдет это только после того, как программа станет сама правильно переводить более 90% фраз. Пока наши достижения значительно скромнее.

Однако это не значит, что специалисты по интерфейсу не могут себя проявить в нашем проекте. Помимо самой программы перевода, у нас есть множество программ, которые не идут пользователю, но которые активно используются разработчиками. В первую очередь это различные утилиты, котрые помогают работать со словарями и с описанием грамматики.

Эти программы могут быть написаны на чем угодно, они должны быть макимально удобны, ну и вообще - "программы для себя"и "одноразовые программы"это целая специальная и очень интересная область.