ЛАБОРАТОРНАЯ РАБОТА №2

Курс «Технологии разработки программного обеспечения»

Тема: Проектирование и макетирование программного продукта.

Цель: Научиться проектировать простейшие системы и составлять документацию по проектированию программного продукта.

Задание:

1. Спроектировать общую архитектуру приложения на основе спецификаций требований, составленных в лабораторной работе №1.
2. Разработать макет интерфейса программного продукта для демонстрации заказчику. Проанализировать и учесть его замечания и пожелания.
3. Оформить разделы документа SDD, относящиеся к предварительному проектированию (приложение 2.1).
4. Программно реализовать первую версию спроектированной системы (создать программные модули и закодировать основные классы / структуры и связи между ними).
5. Оформить отчет, включающий ответы на контрольные вопросы, программный код системы и первую версию SDD-документа.

Контрольные вопросы:

1. В чем отличие предварительного проектирования от детального?
2. Перечислите и кратко опишите архитектурные системные паттерны.
3. Перечислите и кратко опишите паттерны управления.
4. Что такое «связность модуля»? Перечислите и кратко опишите типы связности модуля с указанием значения силы связности для каждого типа.
5. Что такое «сцепление модулей»? Перечислите и кратко опишите типы сцепления модулей.
6. Опишите следующие фундаментальные паттерны проектирования: делегирование, неизменяемый объект, интерфейс, MVC.

Рекомендуемые источники.

1. Брауде Э. Технология разработки программного обеспечения. – СПб.: Питер, 2004. – 655с., ил.
2. Орлов С. Технологии разработки программного обеспечения. – СПб.: Питер, 2002. – 464с.: ил.
3. Константайн Л., Локвуд Л. Разработка программного обеспечения. – СПб.: Питер, 2004. – 592с., ил.
4. Мартин Р. Чистый код: Создание, анализ и рефакторинг. Библиотека программиста. – СПб.: Питер, 2010. – 464с.: ил.
5. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.. Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб.: Питер, 2001. – 368с.: ил.

Приложение 2.1. Фрагменты SDD.

ОПИСАНИЕ ОСОБЕННОСТЕЙ РЕАЛИЗАЦИИ КЛИЕНТСКОГО МОДУЛЯ СИСТЕМЫ АВТОМАТИЗИРОВАННОГО СБОРА ДАННЫХ

ВЕРСИЯ 1.30

СОДЕРЖАНИЕ

ПРИМЕЧАНИЯ

Здесь могут быть примечания по особенностям форматирования кода, именования переменных и функций, использования комментариев и инструментария документации, возможным направлениям рефакторинга (TODO) и т.д.

СТРУКТУРА ПРОЕКТА КЛИЕНТСКОГО МОДУЛЯ

- **логика приложения (классы и соответствующие файлы .h и .cpp)**:

CScanUnit	Единица сканирования
CBarCodeGuide	Справочник ШК
CBarCodeScanner	Сканер ШК
CLogger	Логгер (класс, работающий с логом событий и действий пользователя)
CClientSettings	Настройки и данные текущего клиента и пользователя
…	…

- **окна и GUI-элементы (классы и соответствующие файлы .h и .cpp):**

CAuthorizeDialog	Окно авторизации пользователя; агрегирует класс списка пользователей CUserList; обращается к свойствам CClientSettings
CRequestTasksDialog	Окно получения заданий
CScanDialog	Специальное окно ожидания сканирования ШК. Обрабатывает 3 варианта сканирований: 1) сканирование ШК ячейки для перехода в нее при инвентаризации 2) сканирование ШК упаковки при ее создании в отгрузке 3) сканирование ШК излишков в отгрузках
CCellsDialog	Окно со списком ячеек в заданиях инвентаризации и прибытия; в случае задания прибытия **доступно сканирование**
CLogDialog	Окно с логом событий в определенном контексте: 1) либо все сканирования товара в текущем задании; 2) либо все сканирования товара в режиме перемещения/замены упаковки. Окно лога вызывается из соответствующих окон. Отсутствует в клиентах терминалов.
CServerSettingsDialog	Окно для конфигурации настроек связи с сервером
CImageTextButton	Класс кнопки с картинкой и текстом (в MFC необходимо создавать для такого вида отдельный подкласс стандартной кнопки)
…	…

- **служебные файлы приложения:**

resource.h
stdafx.h и stdafx.cpp
ClientBCSApp.h и ClientBCSApp.cpp

Общая схема классов уровня логики приложения приведена на Рис.1.

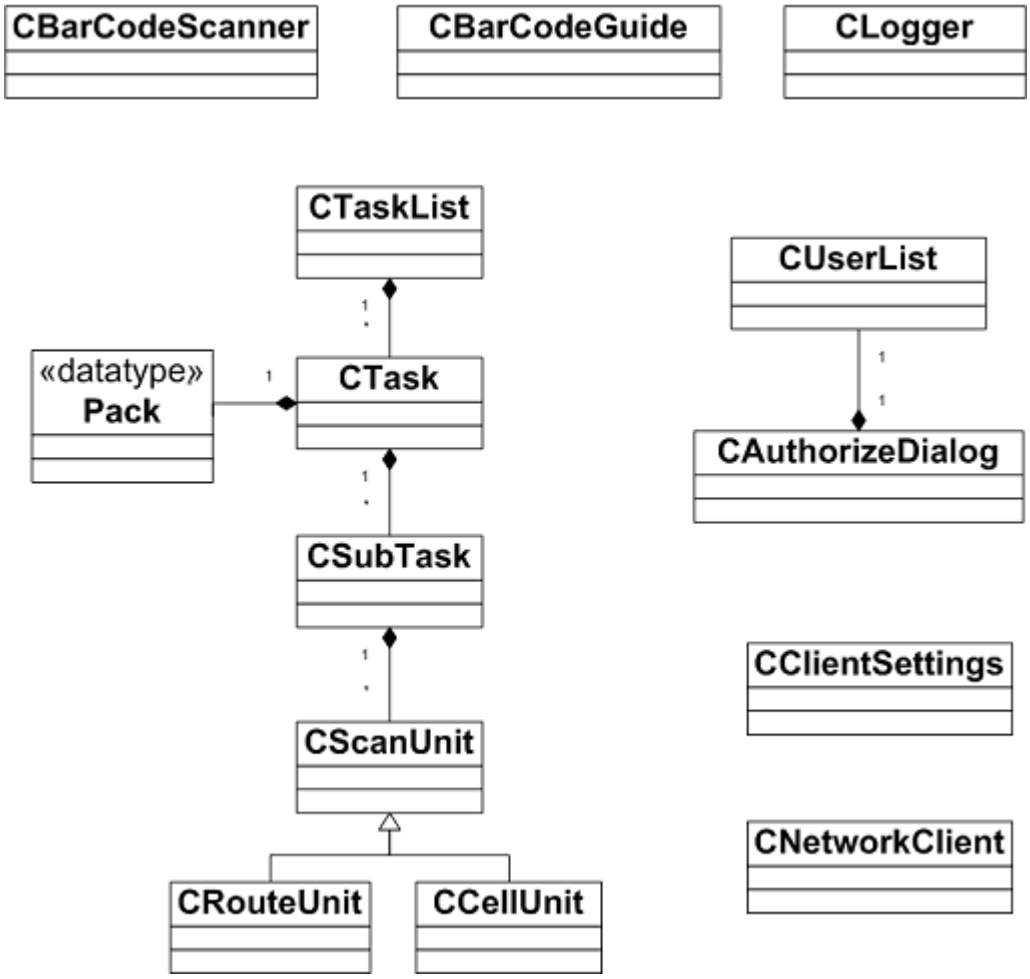

Рисунок 1 – UML-диаграмма классов уровня логики приложения

ОСОБЕННОСТИ РЕАЛИЗАЦИИ И ИСПОЛЬЗОВАНИЯ КЛАССОВ

Этот раздел может быть сформирован с помощью генераторов документации (см. лаб. раб. №5).

- **CClientSettings**

…

- **CLogger**

Класс для логирования всех событий и действий пользователя, для которых разработана приведенная ниже система кодов (от 0 до 16).

```
/*=============== Коды записей в логе ==================================*/

// ------------------------------------------------ Login / Logout
#define LR_USER_LOGIN                   0
#define LR_USER_LOGOUT                  1

// ------------------------------------------------ Scanning
#define LR_SCAN_UNIT                    2
#define LR_SCAN_SURPLUS                 3
#define LR_SCAN_PACK                    4
#define LR_SCAN_CREATE_PACK             5
#define LR_SCAN_REPLACE_PACK            6
#define LR_SCAN_MOVE_PACK               7
#define LR_SCAN_CELL                    8

// ------------------------------------------------ Choosing ("GoTo")
#define LR_CHOOSE_PACK                  9
#define LR_CHOOSE_TASK                  10
#define LR_CHOOSE_SUBTASK               11
#define LR_CHOOSE_CELL                  12

// ------------------------------------------------ Completing
#define LR_COMPLETE_SUBTASK             13
#define LR_SEND_SUBTASK                 14

// ------------------------------------------------ Updating
#define LR_UPDATE_GUIDE                 15
#define LR_UPDATE_TASKS                 16

/*======================================================================*/
```

Функции логгера можно разделить на:

- пишущие (нужны для записи в лог различных событий и действий пользователя)

- читающие (нужны для просмотра лога по сканам в конкретных подзаданиях (FindFirstScanRecord и FindNextScanRecord) и по сканам, сделанным при перемещении товара в упаковках (FindFirstPackRecord и FindNextScanRecord)).

Структура лога была разработана таким образом, чтобы она была сама по себе экономна по памяти и минимально дублировала информацию, которая уже есть в файле .tsk, описывающем структуру заданий на данном клиенте.

В зависимости от типа события, функция и формат записи в лог отличаются:

- ✓ Сканирование ШК: WriteScanOperation() : <тип_операции> <ШК> <Кратность>
- ✓ Переход к задаче/подзадаче/упаковке: WriteChoosingOperation() : <тип_операции> <Название задачи/подзадачи/номер упаковки>
- ✓ Завершение задачи/подзадачи: WriteCompleteOperation() : <тип_операции> <название задачи/подзадачи>

```
class CLogger
{
public:
    CLogger(void);
    ~CLogger(void);

protected:
    FILE *pLogFile;                     // указатель на файл лога
    char cFileName[ 255 ];              // полное имя файла лога

    unsigned int nOffset;               // смещение текущей записи лога относительно начала
    unsigned int nPackOffset;
    unsigned int nPackNextOffset;

    void WriteTime();                           // записать время события
    void WriteString( CString cRecord );        // записать строку события
    CString TimeString( FILETIME& ftime );      // получить форматированную строку с
                                                // датой и временем события
public:
    void OpenForWriting();                      // открыть для записи
    void OpenForReading( bool bFromBeginning ); // открыть для чтения
    void Close();                               // закрыть

    // пишущие функции логгера
    void StartLog( CString cFile );
    void StartPackOperation();
    void WriteScanOperation( unsigned char nScanMode, CString cBarCode, short nQty );
    void WriteChoosingOperation( unsigned char nChooseMode, CString cName );
    void WriteCompleteOperation( unsigned char nCompleteMode, CString cName );
    void WriteGuideOperation();
    void EndLog();

    // читающие функции логгера
    CString FindFirstScanRecord( CString cSubtaskName );
    CString FindNextScanRecord( CString cSubtaskName );
    CString FindFirstPackRecord( );
    CString FindNextPackRecord( );

};
```

Примечание. Строки конвертируются в массив байт (символы - не Юникод, а ANSI (с поддержкой русских символов)). Запись строк происходит в методе WriteString(), где используется функция WideCharToMultiByte() с параметром (CP_ACP).

TODO: рефакторинг – извлечь 2 класса LogWriter и LogReader.

ОПИСАНИЕ CONTROL FLOW ОСНОВНЫХ ЧАСТЕЙ ПРОЕКТА

…

Логирование

Начало сеанса логирования (параллельно с лог-ином юзера):

```
CMainWindow::OnInitDialog()
{
	…
	Logger->StartLog( CClientSettings::sPath + CClientSettings::cLogin + L".log" );
	…
}
```

Примеры операций логирования (в коде проекта присутствуют в обработчиках всех событий, которые описаны в классе логгера (сканирование, переход к задаче и т.д.)):

```
Logger->WriteScanOperation( LR_SCAN_UNIT, cBarCode, guide->getRepetitionByBarCode(cBarCode) );

Logger->WriteChoosingOperation( LR_CHOOSE_TASK, curTask->getTaskName() );

Logger->WriteCompleteOperation( LR_COMPLETE_TASK, curTask->getTaskName() );
```

Завершение сеанса логирования (параллельно с лог-аутом юзера и закрытием главного окна приложения):

```
CMainWindow::OnDestroy()
{
	…
	Logger->EndLog();
	…
}
```

Обновление справочника с сервера

Вызов метода guide.Update() производится в следующих методах:

```
void CMainWindowDialog::OnBnClickedGuide()
void CRouteContentDialog::ProcessUnknownContent( CString cBarCode )
void CCellContentDialog::ProcessUnknownContent( CString cBarCode )
```

…

ФОРМАТЫ БИНАРНЫХ ФАЙЛОВ

…

Файл terminal.dat

- 4 байта : ID терминала (unsigned int)
- 4 байта : количество символов в IP-адресе сервера M (unsigned int)
- M*2 байт : M символов в IP-адресе сервера (M*sizeof(TCHAR))
- 4 байта : порт (int)
- 4 байта : количество IP-адресов сервера в списке N (unsigned long)
- **N блоков (*i*=0,1,2,..,N-1) со структурой:**
  - 4 байта : количество P символов в *i*-ом IP-адресе сервера (unsigned int)
  - P*2 байт : P символов в *i*-ом IP-адресе сервера (P*sizeof(TCHAR))

…

Файл <*user_login*>.log

Файл лога содержит произвольные комбинации из блоков 3 типов.

Блок типа 1 (запись о сканировании):
- 8 байт : Дата / время события (FILETIME)
- **1 байт : тип операции (сканирования) (unsigned char)**
- 1 байт : M – количество символов в строке ШК (unsigned char)
- M байт : M символов в строке ШК (M*sizeof(char))
- 2 байта : кратность отсканированного ШК (short)

Блок типа 2 (запись о переходе в задачу / подзадачу / упаковку / ячейку):
- 8 байт : Дата / время события (FILETIME)
- **1 байт : тип операции (перехода) (unsigned char)**
- 1 байт : M – количество символов в строке названия задачи / подзадачи / упаковки / ячейки (unsigned char)
- M байт : M символов в строке названия задачи / подзадачи / упаковки / ячейки (M*sizeof(char))

Блок типа 3 (запись о завершении задачи / подзадачи):
- 8 байт : Дата / время события (FILETIME)
- **1 байт : тип операции (завершения) (unsigned char)**
- 1 байт : M – количество символов в строке названия завершенной задачи / подзадачи (unsigned char)
- M байт : M символов в строке названия завершенной задачи / подзадачи (M*sizeof(char))