Разработочная подсистема — упорядочивание сопутствующих процессу разработки материалов

Темиргалеев Е. Э.
(Т.Е.Э. 21 авг 2012 г. 13:50:14)

В системе Блэкбокс используются глобальные соглашения о структуре папок (расположении) и именовании файлов, в которых хранятся "части" компонентов. Они довольно просты и базируются на правиле — все составляющие компонента (кодовые и символьные файлы, документация, ресурсы) хранятся в одном вместе. Благодаря этому достаточно легко реализовать программный доступ к такой информации непосредственно (не централизованно) из потребных в ней компонентов (в основном, средства разработки), а добавление компонента в систему или его удаление фактически сводятся просто к добавлению или удалению папки.

Данный подход (на базе систематических соглашений) положительно сказывается на простоте разработки, сопровождения и эксплуатации компонентного ПО (суть компонентов) для всех пользователей системы Блэкбокс.

Процесс разработки сопряжён с появлением сопутствующих материалов, таких как: схемы внутреннего устройства компонента, тестовые модули и тестовые наборы данных, инструментальные документы и т. п. Каких-либо общих соглашений на счёт таких материалов нет, каждый действует как удобнее. Поскольку в контексте разработки материалы непосредственно относятся к компоненту, распространена ситуация, когда разработчики размещают их естественным образом — "поблизости" — среди составляющих компонента. Конечному же пользователю в таких материалах обычно потребности нет, поэтому смешивание разработочных материалов и составляющих компонента в итоге приводит либо к неудобству для разработчика, который должен "чистить" компонент перед упаковкой клиенту, либо к неудобству клиента, который получает балласт, затрудняющий ориентацию среди необходимой информации.

В данной работе сделана попытка кратко и точно изложить соглашения Блэкбокс о размещении файлов компонента и предложена аналогичная схема для организации файлов разработочных материалов — с целью распространения на них положительного опыта систематических соглашений.

1. Репозиторий Блэкбокс — соглашения и терминология

Минимальным компонентом Блэкбокс является модуль Компонентного Паскаля [2, 4.1] (понятие компонента — см. в [2]). *Подсистема* (*subsystem*) — коллекция взаимосвязаных (related) компонентов, размещённых в отдельных подпапках папки Блэкбокс; вся коллекция подсистем — *репозиторий* Блэкбокс (BlackBox *repository*) [1, 3.2].

Например, подсистема *Dev* содержит компоненты — средства разработки. Компилятор Компонентного Паскаля — пример сложного компонента из нескольких модулей; большинство прочих инструментальных компонентов реализованы одномодульно. Один из таких компонентов — *DevRBrowser* — обеспечивает обзор репозитория Блэкбокс, формируя гипертекст со списками подсистем и их содержимого (документация, исходные, кодовые, символьные файлы модулей и

проч.).

Для построения обзора *DevRBrowser* использует упомянутые выше соглашения, изложенные в документации Блэкбокс [1, 3.2]. При этом имеются определённые моменты, которые для общих соглашений не важны, но должны быть разрешены для формирования списка подсистем по содержимому папки Блэкбокс, и в некоторых других случаях. Ниже приведены уточнённые соглашения, детали которых выписаны по реализации компонентов (в основном — *DevRBrowser*), входящих в поставку Блэкбокс версии 1.6-rc6 для Windows.

Соглашение об именах папок подсистем и идентификаторах модулей:

- идентификатор модуля должен начинаться с имени папки подсистемы, заканчивающегося на строчную букву, цифру или подчёркивание, после которых идёт имя модуля в подсистеме;
- если идентификатор модуля нельзя поделить на две части по указанной выше схеме, считается, что модуль не относится к какой-либо подсистеме (скажем — "внеподсистемый").

С точки зрения грамматики Компонентного Паскаля, имя модуля должно быть идентификатором [3, 3]:

ИдентификаторМодуля = ident.

С точки зрения соглашений, идентификатор модуля классифицируется следующим образом:

ИдентификаторМодуля =
= ИмяПодсистемы ИмяМодуляВПодсистеме | ИмяВнеподсистемногоМодуля.
ИмяПодсистемы = {З} С {С}.
ИмяМодуляВПодсистеме = З { С | З }.
ИмяВнеподсистемногоМодуля = (З {З} | С) {С}.
З = ЗаглавнаяБуква.
С = СтрочнаяБуква | Цифра | "_".

Соглашения о структуре папки подсистемы *Sub*:

(1) Подпапка *Mod* содержит (только) файлы документов с исходными текстами модулей. ИмяФайла = ИмяМодуляВПодсистеме. Тип = *Files.docType*.

(2) Подпапка *Code* содержит (только) кодовые файлы модулей. ИмяФайла = ИмяМодуляВПодсистеме. Тип = *Files.objType*.

(3) Подпапка *Sym* содержит (только) символьные файлы модулей. ИмяФайла = ИмяМодуляВПодсистеме. Тип = *Files.symType*.

(4) Подпапка *Docu* содержит файлы документации — модулей и прочую (общую, ...).

(4.1) Имена файлов прочей документации содержат в имени "-" (например: "Sys-Map", "User-Man", "Dev-Man").

(4.2) Остальные файлы — документация модулей. ИмяФайла = ИмяМодуляВПодсистеме. Тип = *Files.docType*.

(5) Подпапка *Rsrc* содержит ресурсные файлы подсистемы (тип — любой):

(5.1) Имя ресурсных файлов модулей рекомендуется начинать с имени модуля:
ИмяФайла = ИмяМодуляВПодсистеме[Суффикс].
Суффикс — в общем произвольная цепочка литер; есть практика использовать натуральные числа, если ресурсов модуля более одного.

(5.2) Прочие файлы — общие, не относятся к модулям.

(5.3) Исключения (по именованию — могут быть отнесены к ресурсным файлам модулей, но никогда таковыми не являются):
- Strings.odc — файл текстовых ресурсов подсистемы;
- Menus.odc — файл меню подсистемы.

Организация файлов "внеподсистемных" модулей. Осуществляется по аналогичной структуре, которая хранится двухуровнево:

(S1) в специальной подсистеме (подпапке) *System*;

(S2) в папке Блэкбокс.

Содержимое первого перекрывается вторым:

- Если подходить формально к выделению имени папки подсистемы из имени модуля, папка

подсистемы — пустая цепочка (""). Доступ к составляющим компонента подсистемы "" по соглашениям о подсистеме посредством *Files* естественным образом даёт (S2).

- Запись должна осуществляться только в (S2); попытка чтения — из (S2), и, затем из (S1), если первая не успешна.

Этот механизм предусмотрен для защиты компонентов библиотеки и каркаса Блэкбокс, которые как раз относятся к категории "внеподсистемных" — *Math*, *Files*, *Stores*, *Models*, ... Стандартная реализация компонентов находится в *System* и заменить её можно только явно, вручную перемещая файлы из (S2) в (S1); средства разработки, следуя соглашениям, этого сделать не могут, т. е. возможность по неосторожности испортить ключевые компоненты системы Блэкбокс, приведя её в нерабочее состояние, исключена.

Итого:

— Модуль можно считать существующим по наличию хотя бы одного из файлов (1)-(4).

— Файл ресурса же можно считать ресурсом модуля (5.1), если есть подходящий по имени модуль.

— Файлы папок (1)-(4), не вписывающиеся в соответствующие соглашения, целесообразно игнорировать для (1)-(3) и отнести к прочей документации для (4).

— Папку можно считать папкой подсистемы, если её имя является именем подсистемы и она содержит хотя бы одну из подпапок (1)-(5).

2. Разработочная подсистема — дополнение к основным соглашениям

Дополнение заключается в следовании основополагающему принципу — держать все составляющие компонента в одном месте, и выражается в размещении всех материалов, исключительно относящихся к процессу разработки подсистемы ***Sub***, в отдельной *разработочной подсистеме* ***Sub***_*dev*. Дополнения к соглашениям о структуре папки подсистемы для папки разработочной подсистемы:

(1) Подпапка *Mod* содержит файлы документов с исходными текстами тестовых модулей.

(4) Подпапка *Docu* содержит файлы разработочной документации модулей и прочими разработочными документами, например, инструментальными.

На каждый модуль основной подсистемы может приходиться несколько тестовых модулей или файлов документации, правила их именования определены аналогично именованию ресурсных файлов:

Имя = ИмяМодуляВПодсистеме["_" Номер].

Например:

Sub/Mod/Name ->

-> Sub_dev/Mod/Name, Sub_dev/Mod/Name_1, ...

-> Sub_dev/Docu/Name, Sub_dev/Docu/Name_1, Sub_dev/Docu/Name_2, ...

На "внеподсистемные" модули дополнение распространяется формально. Считая папку Блэкбокс папкой подсистемы "", получаем: разработочные материалы "внеподсистемных" модулей хранятся в папках ***System***_*dev* (S1) и _*dev* (S2); материалы в (S2) перекрывают материалы в (S1). Хотя назначение механизма перекрытия для *System* теряет смысл применительно к разработочным материалам, формальный подход с сохранением перекрытия оправдан тем, что не требует введения в соглашения особенных для разработочных материалов исключений. Последнее позволяет сохранить единообразность машинной обработки.

Итого:

— разработочные материалы подсистемы ***Sub*** хранятся в одной *отдельной папке* разработочной подсистемы ***Sub***_*dev*, играющей роль "строительных лесов";

— установлено соответствие между модулем подсистемы ***SubMod*** и его тестовыми модулями ***Sub***_*dev****Mod***, ***Sub***_*dev****Mod***_*1*, ..., а также, аналогично — разработочной документацией.

Более детальное регламентирование (соглашения по оформлению разработочной документации, обязательность её написания, разработку тестов и проч.), по мнению автора, необходимо для систематического процесса разработки. С другой стороны, сложно предложить какие-либо глобальные соглашения, поскольку чем больше детализация, тем больше и зависимость

оптимального выбора от конкретного коллектива, специфики решаемых им задач и т. п. Некоторые соображения на эту тему приведены в следующем пункте в виде примеров.

3. Примеры разработочных материалов и их оформления

3.1 Документация внутреннего интерфейса модуля

В Блэкбокс приведены соглашения [4] по оформлению документации интерфейса модуля. В них идёт речь об описании всех экспортируемых элементов модуля. Однако, с технической точки зрения, интерфейс модуля не всегда полностью входит в интерфейс экспорта компонента, т. е. концептуально распадается на:

- внешний — "неизменяемые" [2, 4.1] элементы интерфейса экспорта, документированные для разработчиков клиентских модулей (компонентов);
- внутренний — элементы помеченные как "для внутреннего использования" (used internally), соответственно другими модулями данного компонента или другими компонентами от того же разработчика, которые поставляются одним комплектом.

Kernel, ядро системы Блэкбокс, — пример модуля, интерфейс которого полностью внутренний. Интерфейс модуля *DevDebug* содержит как внутренние, так и внешние элементы.

Документация внутреннего интерфейса — пример разработочной документации модуля. Для её оформления целесообразно использовать соглашения [4].

3.2 Документация реализации (полная документация) модуля

Описание внутренних, не экспортированных элементов (всех элементов) модуля. Оформляется аналогично соглашениям [4], с сохранением меток экспорта.

Документация реализации помогает разработчику ориентироваться в модуле, особенно при смене разработчика или возобновлении работ после паузы. Поскольку содержит пред- и пост-условия, инварианты, — способствует доказательному программированию. Когда пишется параллельно модулю, что наиболее предпочтительно, выступает как объективный фактор, сдерживающий на противоходе "клепательский" полёт мысли: необходимость осмысления только что добавленных в модуль элементов для фиксирования их описания письменно стимулирует поиск более простого решения, которое облегчит работу по документированию путём избавления от новых элементов или сокращения их числа.

3.3 Инструментальные документы с наборами тестирующих команд

Наиболее просто тестирование организуется для процедур-команд, чтение параметров которых реализует подход "текст как интерфейс". Достаточно сделать документ с тестовыми наборами данных. При этом наиболее удобен случай, когда команда допускает активацию коммандером (*DevCommanders*), т. е. "умеет" считывать параметры с отрезка текста *DevCommanders.par*. Этот отрезок при надлежащем оформлении явно виден, команды активируется одним щелчком мыши; специальная команда *ert0devCommanders.Exec* позволяет одним щелчком активировать последовательность команд. Документ с тестовыми наборами данных оказывается инструментальным документом с набором тестирующих команд:

ert0devCommanders.Exec
Модуль не найден
DevDebug.UnloadThis A B ◢
Модуль не выгружаемый (есть клиенты)
DevDebug.UnloadThis TextModels TextViews ◢
Выгрузка
ObxHello0.Do — *выполнение команды гарантирует загрузку ObxHello0*

Если команда читает данные из других "мест", для тестирования крайне желательно организовать её вызов единообразно — посредством ориентированного на массовую активацию коммандера. Например, для команд, читающих из выделенного текста, это делает команда-обёртка *ert0devCommanders.SelectAndDo*:

ert0devCommanders.Exec
"ert0devCommanders.SelectAndDo('ObxRatCalc.Simplify')" 1/2 - 1/3
"ert0devCommanders.SelectAndDo('ObxRatCalc.Approximate')" 1/2 - 1/3
"ert0devCommanders.SelectAndDo('ObxRatCalc.Approximate')" 3/9

Приведённые примеры инструментальных документов представляют ещё один вид разработочной документации — как модуля, так и подсистемы.

3.4 Тестовые модули

Общее решение "коммандеризации" вызова произвольных команд и процедур, не являющихся командами, заключается в написании специальных тестовых команд — одноимённых процедур тестового модуля.

Ниже показано, как мог бы выглядеть типовой тестовый модуль для модуля *Integers* компонентной библиотеки Блэкбокс, реализующего целочисленную арифметику неограниченной точности:

"ert0devCommanders.SelectAndDo('DevCompiler.CompileSelection')"

```
MODULE System_devIntegers;

   IMPORT Log, In := i21sysIn, X := Integers;

   PROCEDURE LogInteger (x: X.Integer);
      VAR    s: ARRAY 4096 OF CHAR;
   BEGIN
      X.ConvertToString(x, s); Log.String(s)
   END LogInteger;

   PROCEDURE InInteger (OUT x: X.Integer);
      VAR    s: ARRAY 4096 OF CHAR;
   BEGIN
      In.String(s); IF In.done THEN X.ConvertFromString(s, x) END
   END InInteger;

   PROCEDURE Entier*;
      VAR    x: REAL; y: X.Integer;
   BEGIN
      In.Open; In.Real(x);
      WHILE In.done DO
         y := X.Entier(x); LogInteger(y); Log.Ln;
         In.Real(x)
      END
   END Entier;

   PROCEDURE Product*;
      VAR    a, b, c: X.Integer;
   BEGIN
      In.Open; InInteger(a); InInteger(b);
      WHILE In.done DO
         c := X.Product(a, b); LogInteger(c); Log.Ln;
         InInteger(a); InInteger(b)
      END
   END Product;
   …
END System_devIntegers.
```

Примеры тестирующих команд:

```
System_devIntegers.Entier   -1.0   0.5E10 0.5E50   0.5E51
System_devIntegers.Product
  "10"   "-9"
  "-12345678987654321"   "-98765432123456789"
```

Обычно удобным оказывается размещать их в одном документе с исходником тестового модуля.

Для тестирования внутренних элементов модуля *X*, их нужно делать экспортированными на время тестирования. Автоматизировать этот процесс можно при помощи переключаемых меток экспорта — спецвьюшек *ert0devSpecViews.View* в режиме *ert0devSpecViews.exportMark*, реализующих специальный (ориентированный на задачу) вид условной компиляции. Тестовый модуль внутренних элементов стоит делать отдельно от модуля для тестирования элементов интерфейса модуля, чтобы последний не зависел от режима экспорта спецвьюшки.

Компиляция приведённого ниже примера тестового модуля внутренних элементов требует пометки спецвьюшкой элементов *Index* и *GetLength* модуля *Integers*:

```
MODULE Integers;
  …
  TYPE
    Index = INTEGER;
  …

  (* Data Format Support *)
  …
  PROCEDURE GetLength (x: Integer; OUT len: Index; OUT negative: BOOLEAN);
  …

MODULE System_devIntegers_1;

  IMPORT Log, In := i21sysIn, TextMappers, X := Integers;

  PROCEDURE GetLength*;
    VAR   x: X.Integer; len: X.Index; negative: BOOLEAN;
  BEGIN
    In.Open; In.scanner.Scan;
    WHILE In.scanner.type IN {TextMappers.int, TextMappers.real, TextMappers.string} DO
      CASE In.scanner.type OF
      | TextMappers.int:   x := X.Long(In.scanner.int)
      | TextMappers.real:   x := X.Entier(In.scanner.real)
      | TextMappers.string:   X.ConvertFromString(In.scanner.string, x)
      END;
      X.GetLength(x, len, negative);
      Log.Int(len); Log.Tab; Log.Bool(negative); Log.Ln;
      In.scanner.Scan
    END
  END GetLength;
  …
END System_devIntegers_1.

ert0devSpecViewsCmds.ExportOn
DevCompiler.CompileThis Integers System_devIntegers_1

System_devIntegers_1.GetLength
```

Список литературы

1. Документация Блэкбокс 1.6-rc6. BlackBox Tutorial, 3 BlackBox Design Practices.
2. Пфистер К. Компонентное программное обеспечение // oberoncore.ru: сайт, 2012. URL: http://oberoncore.ru/library/component_soft (дата обращения: 15.08.2012).
3. Документация Блэкбокс 1.6-rc6. Component Pascal Language Report.

4. Документация Блэкбокс 1.6-гс6. Documentation Conventions.

Приложение: набор изменений Блэкбокс для показа разработочных документов модуля

(в качестве примера)

В *DevReferences* к командам показа документации и исходного текста модуля, имя которого выделено в тексте, добавлены команды показа для этого же модуля документов с исходником тестового модуля и разработочной документацией. Параметр *no* процедуры-команды — неотрицательный номер документа.

MODULE DevReferences;⇨

```
IMPORT …, Strings;

VAR    devModNo: ARRAY 16 OF CHAR;
…
PROCEDURE Show (category: Files.Name);
   VAR c: TextControllers.Controller; s: TextMappers.Scanner; beg, end: INTEGER;
      module, ident: TextMappers.String;
BEGIN
   c := TextControllers.Focus();
   IF c # NIL THEN
      IF c.HasSelection() THEN
         c.GetSelection(beg, end);
         s.ConnectTo(c.text); s.SetOpts({7});    (* acceptUnderscores *)
         s.SetPos(beg); s.Scan;
         IF s.type = TextMappers.string THEN
            module := s.string; ident := "";
            CheckModName(module, c.text);
            s.Scan;
            IF (s.type = TextMappers.char) & (s.char = ".") THEN
               s.Scan;
               IF s.type = TextMappers.string THEN ident := s.string END
            END;
            ShowText(module + devModNo, ident, category)
         ELSE Dialog.ShowMsg("#System:WrongSelection")
         END
      ELSE Dialog.ShowMsg("#System:NoSelection")
      END
   ELSE Dialog.ShowMsg("#System:NoFocus")
   END
END Show;
…
PROCEDURE ShowDev (IN cat: ARRAY OF CHAR; no: INTEGER);
   VAR
BEGIN
   ASSERT(0 <= no , 20);
   IF no = 0 THEN
      devModNo := ""
   ELSE
      Strings.IntToString(no, devModNo);
      devModNo := "_" + devModNo
   END;
   Show("Dev" + cat); devModNo := ""    (* считаем  что трапов нету  *)
END ShowDev;

PROCEDURE ShowDevMod* (no: INTEGER);
BEGIN
   ShowDev("Mod", no)
END ShowDevMod;
```

```
    PROCEDURE ShowDevDocu* (no: INTEGER);
    BEGIN
        ShowDev("Docu", no)
    END ShowDevDocu;

END DevReferences.⇦
```

Модернизация *StdDialog* — для поддержки изменений *DevReferences*.

```
MODULE StdDialog;⇨
    …
    PROCEDURE GetSubLoc* (mod: ARRAY OF CHAR; cat: Files.Name;
                                OUT loc: Files.Locator; OUT name: Files.Name);
        VAR sub: Files.Name; file: Files.File; type: Files.Type; dev: BOOLEAN; i: INTEGER;
    BEGIN
        IF (cat[0] = "S") & (cat[1] = "y") & (cat[2] = "m") THEN type := Kernel.symType
        ELSIF (cat[0] = "C") & (cat[1] = "o") & (cat[2] = "d") & (cat[3] = "e") THEN type := Kernel.objType
        ELSE type := ""
        END;
        Kernel.SplitName(mod, sub, name); Kernel.MakeFileName(name, type);
        dev := (cat[0] = "D") & (cat[1] = "e") & (cat[2] = "v");
        IF dev THEN
            i := 3; WHILE cat[i] # 0X DO cat[i - 3] := cat[i]; INC(i) END;
            cat[i - 3] := 0X
        END;
        IF dev THEN loc := Files.dir.This(sub + "_dev") ELSE loc := Files.dir.This(sub) END; file := NIL;
        IF loc # NIL THEN
            loc := loc.This(cat);
            IF sub = "" THEN
                IF loc # NIL THEN
                    file := Files.dir.Old(loc, name, Files.shared);
                    IF file = NIL THEN loc := NIL END
                END;
                IF loc = NIL THEN
                    IF dev THEN loc := Files.dir.This("System_dev") ELSE loc := Files.dir.This("System") END;
                    IF loc # NIL THEN loc := loc.This(cat) END
                END
            END
        END
    END GetSubLoc;
    …
END StdDialog.⇦
```

Добавление команд открытия разработочных документов в меню "Info" и контекстное меню текста. Соответствие между номерами разработочных документов и их содержанием, а также набор команд в меню — предмет внутренних соглашений.

```
Dev/Rsrc/Menus⇨
MENU "Info"
    …
    "&Documentation"    ""    "DevReferences.ShowDocu"    "TextCmds.SelectionGuard"
    "Тестовый (эл-ты интерфейса)"    ""    "DevReferences.ShowDevMod(0)"    "TextCmds.SelectionGuard"
    "Тестовый (внутренние эл-ты)"    ""    "DevReferences.ShowDevMod(1)"    "TextCmds.SelectionGuard"
    "Разр-я док-я (полная)"    ""    "DevReferences.ShowDevDocu(0)"    "TextCmds.SelectionGuard"
    "De&pendencies"    ""    "DevDependencies.Deposit;StdCmds.Open"    "TextCmds.SelectionGuard"
    …
END
⇦
Text/Rsrc/Menus⇨
MENU "*" ("TextViews.View")
    …
    "&Documentation"    ""    "DevReferences.ShowDocu"    "TextCmds.SelectionGuard"
    … тоже …
```

```
    SEPARATOR
    …
END
⇦
```