УДК 007.051

Реализация механизма подключения провайдера в биллинговой системе авиапредприятия, настройка интерфейса взаимодействия с конечным пользователем.

А.Ю. Арженко, С.А. Байраковский, В.А. Вестяк

Аннотация

Целью данной статьи является исследование и реализация одного из способов подключения новых провайдеров в биллинговой системе авиапредприятия, а также реализация “гибкой” настройки интерфейса взаимодействия данного провайдера с конечным пользователем, что особенно важно для оказания услуг регистрации авиаперевозок, реализации системы продажи авиабилетов. В статье описаны инструменты разработки и примеры программного кода, которые можно использовать в качестве практического применения для решения подобной проблемы. Приведены примеры реализации взаимодействия объектов БД с программным модулем, управляющим настройками работы провайдеров.

Ключевые слова

Биллинговая система; провайде; автоматизация; регистрация; база данных; сценарий работы; интерфейс.

На сегодняшний день, одним из немаловажных факторов успешного развития гражданской авиации является повышение лояльности в отношении клиентов. Все чаще авиакомпании вынуждены повышать эффективность сервисов, оказывающих автоматизированные услуги по бронированию авиабилетов, оказанию услуг регистрации авиаперевозок, получению актуальной информации по любому интересующему рейсу в любой точки земного шара. Ключом к реализации таких сервисов в рамках биллинговой системы является возможность быстрой и гибкой регистрации новых нестандартных провайдеров связи.

Мало просто зарегистрировать самого провайдера в системе, как минимум необходимо поддержать специфический интерфейс, через который он будет работать. Для такой цели необходимо реализовать универсальный механизм, цель которого будет заключаться в описании сценария работы провайдера, его регистрации, хранении последовательности интерфейсов, предоставляемых конечному потребителю – клиенту.

Для начала необходимо определиться с инструментами разработки. В качестве СУБД будем использовать Oracle 10g, в качестве языка программирования для реализации служебных приложений будем использовать Delphi.

Саму задачу можно разделить на несколько этапов:

- Определить структуру и зависимость между таблицами, в которых будут храниться данные, необходимые для работоспособности универсального механизма
- 1Необходимо определиться с набором пакетов и функций, которые будут обеспечивать связь между БД и служебными приложениями (добавление/редактирование/удаление объектов настройки универсального механизма).
- Написание служебного приложения, посредством которого можно будет заводить нового провайдера и настраивать сценарий его работы, с учетом специфики данного провайдера (будь то обычный провайдер по оплате сотовой связи или провайдер по оплате ЖД, Авиабилетов).

В качестве основных таблиц можно выделить таблицу хранения данных по провайдеру, туда же

будут записываться все регистрационные данные по провайдеру (пусть таблица называется PROVIDERS). В зависимости от специфики провайдера, как писалось ранее, для взаимодействия с конечным пользователем необходимо выделить ряд “атрибутов интерфейса” (то с чем клиент будет непосредственно взаимодействовать). Например, поле ввода номера телефона или поле выбора авиарейса и т д. Для этой цели необходимо создать таблицу (PROVIDER_FIELDS), в которой, за каждым провайдером будет закреплен набор атрибутов, которые в свою очередь могут быть как входными, так и выходными параметрами. Клиент должен не только уметь вводить необходимую информацию, но и получать ее в тот или иной момент, в зависимости от предполагаемого сценария работы провайдера. Взаимосвязь между таблицами PROVIDERS и PROVIDER_FIELDS будет осуществляться по идентификатору провайдера (PROVIDER_ID).

Структуру таблицы PROVIDER_FIELDS необходимо рассмотреть более подробно. Каждый атрибут провайдера должен содержать минимальный набор конфигураций, необходимый для корректного отображения на терминале.
Минимальный набор конфигураций может выглядеть следующим образом:

- **Идентификатор атрибута** (уникальный в разрезе провайдера);
- **Тип атрибута** (входной, выходной, входной/выходной);
- Маска ввода атрибута на терминале (например (###) ###-##-##, если атрибут является номером телефона, ##-### номер авиарейса и т. д.);
- **Маска печати атрибута на чеке** (практически всегда идентична маски печати атрибута на терминале);
- **Длинна атрибута** (минимальная и максимальная);
- **Регулярное выражение для проверки введенного значения** (пример дня номера телефона: **[\(.]?[2-9]\d\d[\).]?[2-9]\d\d[-.]\d{4}**);

Помимо всех вышеперечисленных конфигураций, каждому атрибуту, отображаемому на терминале, должны быть сопоставлены ряд подписей (заголовок атрибута, подпись, дополнительная информация). Это необходимо для более “дружественного отображения” атрибута на экране терминала.

Проблема хранения информации по провайдеру и его атрибутам решена, однако возникает следующий вопрос: в какой момент времени отображать тот или иной атрибут и в какой последовательности? Ответом на этот вопрос будет являться создание новой таблицы (SCRIPT_SEQUENCE), в которой будут храниться элементы сценария интерфейсов по каждому провайдеру. Взаимосвязь с предыдущими объектами системы, описанными выше, все также осуществляется по идентификатору провайдера (PROVIDER_ID).

Состав основных полей данной таблицы можно рассмотреть более подробно:

- **Номер шага** (фактически таблица SCRIPT_SEQUENCE содержит описания набора интерфейсов в разрезе каждого зарегистрированного провайдера), порядок отображения интерфейса;
- **Действие**

В зависимости от типа действия можно сконфигурировать терминальный софт на совершение той или иной специфической операции.

Допустим, что в нашем случае все типы действий будут описаны в специальном справочнике (SCRIPT_ACTION_TYPE)

1. INFO - Отправка информационного запроса;
2. EDIT - Редактирование параметра платежа;
3. SETFIELD - Установить значение переменной;
4. VIEW - Отобразить информацию на экране;
5. EDITLIST – Редактирование списка;

- **Тип редактора**

  Определяет тип устройства ввода для атрибута провайдера. Например:

  1. GENERIC – Стандартный редактор;
  2. KEYBOARD – ввод данных с экранной клавиатуры;
  3. CHOICE – Выбор из нескольких опций;
  4. BARCODE – Ввод данных со штрих-кода (при условии, что терминал оборудован считывателем штрих-кода);
  5. LISTBOX – Выбор из длинного списка со скроллингом;

- **Наименование параметра**

  В данном поле идет сопоставление идентификатору атрибута провайдера из таблицы PROVIDER_FIELDS;

- **Блокировка** – возможность заблокировать интерфейс в любой момент времени;

- **Конфигурационный файл, условие выполнения**, по сути, являются двумя разными XML документами. Данные конфигурации служат для более гибкого взаимодействия интерфейсов провайдера между собой.

  Пример использования:

Пусть существует провайдер, оплата по которому может производиться на основе считанного штрих-кода или при вводе необходимых данных на терминале. Пример программирования первого интерфейса (конфигурационный файл):

```
<param>
<items>
```

```
<!-- На экране терминала отображается интерфейс, состоящий из двух кнопок, выбор способа ввода данных посредством считывания информации штрих-кода или ввода вручную. -->
<item id="1" text="WBARCODE" />
<item id="0" text="WTOUCHSCREEN" />
</items>
</params>
```

Условие выполнения данного сценария описывается следующим XML-документом:

```
<params>
 <!—Если ввод данных НЕ был осуществлен при помощи штрих кода – применяем ручной способ ввода. -->
 <condition field="@BARCODEREADERPRESENT" op="eq" value="1" />
 <default value="0" />
</params>
```

Далее опишем интерфейсы для ввода номера счета, суммы и периода оплаты, конфигурационных файлов у них не будет, однако будут условные, в условии будет описано то, что если ввод осуществлялся при помощи штрих-кода показывать их не надо, если же нет – необходимо отобразить поля ввода данных.

Условия у 2х интерфейсов будут одни и те же.

```
<params>
 <condition field="INPUT_TYPE" op="eq" value="0" />
</params>
```

После ввода данных терминал может предложить учесть в текущем платеже копейки (как пример, может быть все что угодно). Опишем соответствующий интерфейс для этого, конфигурационный файл:

```
<params>
<items>
<item id="TRUE" text="WPENALTYON" />
```

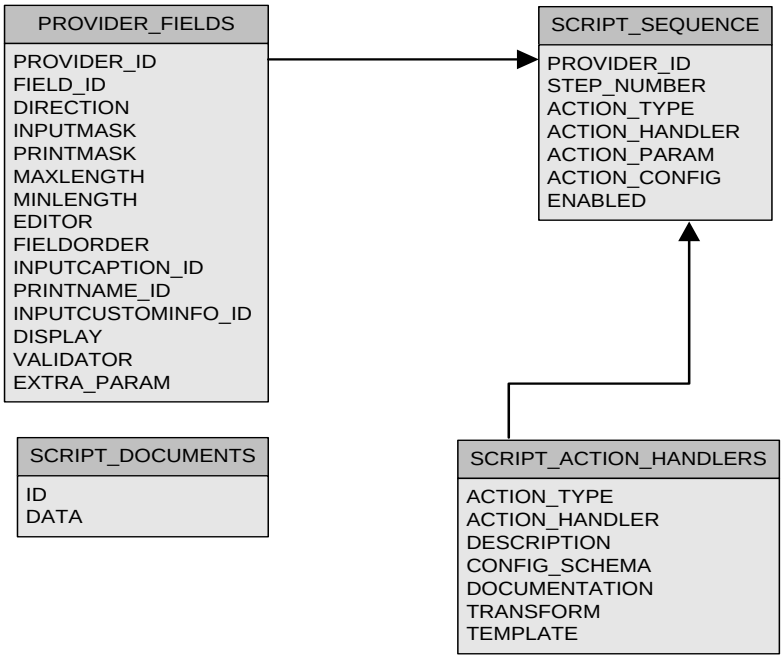
```
<item id="FALSE" text="WPENALTYOFF" />
</items>
</params>
```

Далее опишем последний интерфейс, его отображение будет зависеть, как и прежде от способа ввода информации, в случае если ввод производится со штрих-кода клиенту на экран будут выведены данные о периоде оплаты и номера счета.

```
<params>
 <!-- Входные параметры запроса (любое число параметров),
     где field - имя переменной, tag - наименование тега
     в который будет помещено значение переменной -->
 <input field="BAR_CODE" tag="BAR_CODE" />
 <!-- Выходные параметры запроса (любое число параметров),
     где field - имя переменной в которую сохраняется
     значение, tag - наименование тега из которого будет
     извлекаться значение, type - тип переменной
     (value - значение, list - список), expiration -
     срок жизни сохраняемого значения в секундах (может
     отсутствовать). -->
  <output field="ACCOUNT" tag="ACCOUNT" type="value" />
</params>
```

Таким образом, итогом настройки данных интерфейсов для вымышленного провайдера стал следующий сценарий: терминал предлагает клиенту ввести штрих-код для оплаты услуг вымышленного провайдера, в случае если вводится правильный штрих код - в ответ клиент получает номер счета, сумму и период оплаты, если выбран ручной способ ввода – терминал предлагает ввести все данные вручную. При вводе суммы оплаты терминал дополнительно предлагает учитывать копейки при проведении платежа или нет. Данная конфигурация интерфейсов может подойти, например, для оплаты ЖКХ или простейшей организации продажи авиационных или железнодорожных билетов.

Для большей наглядности можно привести схему взаимодействия всех вышеперечисленных объектов системы:



В данной схеме присутствует объект SCRIPT_DOCUMENTS, назначение данной таблицы несет в себе вспомогательный характер. XML-документы, описывающие конфигурационные файлы и условия выполнения, крайне затруднительны как для просмотра, так и для первоначального запоминания. По сути, документы описывают некие условия или списки объектов.

Для более удобной работы со служебным приложением имеет место введения XSL шаблонов, описывающих структуру вышеперечисленных XML документов. Именно в таблице SCRIPT_DOCUMENTS будут храниться XSL шаблоны.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
 <xsl:output method="text"/>
 <xsl:template match="/params">
  <xsl:for-each select="condition">
   <xsl:if test="position() > 1">
```

```
      <xsl:text> и </xsl:text>
    </xsl:if>
    <xsl:text>(</xsl:text>
    <xsl:value-of select="@field"/>
    <xsl:text> </xsl:text>
    <xsl:choose>
      <xsl:when test="@op = 'gteq' or @op = 'greater-or-equal' or @op = 'greater-than-or-equal'">
        <xsl:text>&gt;=</xsl:text>
       ….
      </xsl:when>
    </xsl:choose>
    <xsl:text> </xsl:text>
    <xsl:value-of select="@value"/>
    <xsl:text>)</xsl:text>
  </xsl:for-each>
  <xsl:if test="default/@value">
    <xsl:text>, по умолчанию </xsl:text>
    <xsl:value-of select="default/@value"/>
  </xsl:if>
 </xsl:template>
</xsl:stylesheet>
```

При сопоставлении данного XSL шаблона и приведенного выше XML документа (конфигурационного файла) получается следующий результат:

```
<!—XML документ -->
<param>
<items>
<item id="1" text="WBARCODE" />
```

```
<item id="0" text="WTOUCHSCREEN" />
</items>
</params>
<!—Результат сопоставления с XSL шаблоном -->
```

Выбор из списка “1”, “0”

Сопоставление с XSL шаблоном производится непосредственно в программе, посредством работы с классом IXMLDOMDocument3, IXSLTemplate, IXMLDOMParseError, IXSLProcesso. Для того чтобы успешно работать с классами, которые в свою очередь имеют прямую зависимость от модуля MSXML2_TLB.

В качестве примера взаимодействия XML документа и XSL шаблона, думаю, будет интересно привести пример функции, на вход подается сам XML документ и XSL шаблон, на выходе – описание XML документа в соответствии с шаблоном:

```
function TfrmFieldsAndScripts.GetXslDesc2(xmlBLOB, xslBLOB : TBlobField)
: string;
var    xslt : IXSLTemplate;
       xmlDoc : IXMLDOMDocument3;
       xslDoc : IXMLDOMDocument3;
       err : IXMLDOMParseError;
       xslProc : IXSLProcessor;
begin
 Result := '';
 xslt := CoXSLTemplate60.Create;
 xslDoc := CoFreeThreadedDOMDocument60.Create;
 xslDoc.async := False;

xslDoc.loadXML(TBlobField(dm.ScriptDocumentsFT.FieldByName('data')).Val
ue);
```

```
 if xslDoc.parseError.errorCode <> 0 then begin
  err := xslDoc.parseError;
  Result := ''; Exit;
 end else begin
  xslt.stylesheet := xslDoc;
  xmlDoc := CoDOMDocument60.Create;
  xmlDoc.async := False;
  xmlDoc.loadXML(xmlBLOB.Value);
  if xmlDoc.parseError.errorCode <> 0 then begin
   err := xmlDoc.parseError;
   Result := '';
   Exit;
  end else begin
   xslProc := xslt.createProcessor();
   xslProc.input := xmlDoc;
   xslProc.transform();
   Result := xslProc.output;
  end;
 end;
end;
```

В данной статье подробно рассмотрен один из механизмов подключения провайдера в биллинговой системе. Приведены примеры XML-документов (шаблонов), описывающих сценарий, по которому работает новый провайдер.

Данную статью можно использовать как практическое пособие, на основе которого можно разрабатывать собственные механизмы регистрации провайдеров в биллинговой системе. Также в статье описан принцип разработки приложения для регулировки настройки провайдера, приведены примеры взаимодействия документов формата XML, XSL в среде

разработки Delphi. В приложениях 1 и 2 продемонстрированы экраны приложения, описываемого авторами в данной статье.

Библиографический список.

1. Большова Г. Минуты любят счет // Сети. 1999. № 10.

2. Coutier P. Charging Forwards// Mobile Communications International. 2003. Nov.
3. Голомшток Л.В. Биллинговые системы для мобильной связи.//Технологии и средства связи. 2003. № 6.
4. Ляхов А.Ю. VoIP - дополнительные услуги и сервисы на последней миле//ИнформКурьерСвязь. 2003. № 6.
5. Faynberg I., Gabuzda L., Hui-Lan Lu. Converged Network and Services. - N.Y.: John Wiley&Sons, 2000.
6. Компания «Восточный ветер» обзорная статья о перспективах развитии биллинговых систем. // http://www.amobile.ru/billing/development.htm

Сведения об авторах

Арженко Александр Юрьевич, профессор Московского авиационного института, (государственного технического университета), дфмн.

МАИ, Волоколамское ш., 4, Москва, А-80, ГСП-3, 125993;
тел: 8 (903) 769-08-27,: 497-94-03

Байраковский Станислав Антонович, старший специалист департамента разработки систем лояльности, ООО “РУСОФТ”, аспирант Московского авиационного института (государственного технического университета) тел: 8 (906) 757-81-13, 313-97-31, E-mail: stanislav@rucard.net

Вестяк Владимир Анатольевич, заведующий кафедрой Московского авиационного института (государственного технического университета, кфмн, доцент.

МАИ, Волоколамское ш., 4, Москва, А-80, ГСП-3, 125993;
E-mail: v.a.vestyak@mail.ru

_____ Арженeнко А.Ю. «___»_____ 2010г.

_____ Байраковский С.А. «___»_____ 2010г.

_____ Вестяк В.А. «___»_____ 2010г.