Технология доступа к данным в Информационном web-портале

А.В. Босов, Р.Б. Чавтараев

Аннотация. Описана технология, обеспечивающая взаимодействие программного комплекса «Информационный web-портал» с внешними информационными источниками. Подробно рассмотрена существующая реализация портала в связи с задачей формирования контента, поставляемого в портал разнородными информационными системами. На примерах проиллюстрированы принципы работы ключевого компонента инфраструктуры – сервера интеграции и доступа.

Введение

Концепция единой точки доступа к распределенным разнородным данным – основа любой портальной технологии. Например, «вертикальным порталом» как раз и называют web-систему, обеспечивающую доступ и манипулирование корпоративной информацией через единую точку входа [1]. Реализуя эту концепцию, разработчики большинства известных решений используют так называемые «адаптеры», обеспечивающие преобразование данных внешних систем в форматы, понятные порталу. Такая, несомненно упрощенная, формулировка отражает суть подхода, детализируемую уже конкретным решением. В общем случае можно считать, что подход порталов к интеграции внешних систем таков [2]: один адаптер – одна внешняя система, внутри же портала информация представляется в некотором унифицированном внутреннем формате. Таким образом, если какому-то портальному приложению требуется работать с определенным источником, то он полностью зависит от адаптера этого источника. В целом такой подход представляется вполне состоятельным, но порождает определенные сложности, попытка преодолеть которые или, по крайней мере, минимизировать их, предложена в Информационном web-портале [3-5]. В связи с тем, что под порталом понимается система, обеспечивающая удобный сервис при работе с множеством самостоятельных информационных источников, образующих распределенную федеративную среду, возможности, предоставляемые порталом в этой части, выходят на первый план.

Предлагаемый подход основан на уточнении понятия «адаптер». Именно, в Информационном web-портале работа адаптера регламентируется требованиями поддержки строго определенного функционального интерфейса, формирование которого базируется на объектно-ориентированной парадигме. Именно, адаптер обеспечивает выполнение строго фиксированного набора команд, задающих все возможные операции над всеми информационными объектами, которые поддерживаются внешней системой. Каждому запросу к внешней системе соответствует в точности одна команда.

Такой подход, с одной стороны, ограничивает возможности взаимодействия с информационным источником. С другой стороны, как показано далее, существенно облегчает реализацию портального решения в целом и расширяет возможности интеграции поставляемого разнородными системами контента. Практически для любой системы, заинтересованной во взаимодействии с порталом, можно

с минимальными затратами разработать адаптер портала, не внося принципиальных изменений в функционал самой системы, и подключить его «на ходу» к порталу.

Набор команд, которые должен реализовывать адаптер, формируется достаточно гибко, позволяя реализовать как самые простые схемы взаимодействия (например, полнотекстовой поиск по ресурсу), так и довольно сложные (вплоть до объединения схем репозиториев источника и портала). Принципиальным для формирования этого набора является требование унифицированно работать с источниками как с хранилищами данных, т.е. посредством функционального интерфейса осуществлять доступ и манипулирование данными внешней системы.

1. Сервер интеграции в архитектуре портала

1.1. Используемые компоненты портальной инфраструктуры

В существующей реализации Информационного web-портала имеется достаточно развитая программная инфраструктура [5], представляющая собой ряд взаимодействующих служб. Рассматриваемая в данной работе служба, обеспечивающая взаимодействие внешних информационных источников, названа *сервером интеграции и доступа*. С работой этого сервера наиболее тесно связаны следующие портальные подсистемы:

- сайт и система управления содержанием;
- подсистема безопасности (прежде всего сервис аутентификации);
- служба поиска.

Сайт портала и *система управления содержанием* в конечном итоге обеспечивают отображение данных и форм манипулирования этими данными пользователю. Этим службам необходимо иметь возможность извлекать данные из поставщика информации. Единственным таким поставщиком является сервер интеграции и доступа. Сами службы «не знают», откуда и как данные извлекаются. Они обращаются, используя единственный унифицированный механизм, к серверу интеграции и доступа, который выполняет запрос и возвращает результат. Причем возвращаемый результат также унифицирован: им всегда является реляционный набор, представляемый объектом DataSet используемой платформы .Net.

Сервис аутентификации предназначен для идентификации пользователей в портале и является основной частью подсистемы безопасности. Этот сервис осуществляет гарантированную идентификацию пользователя и привязку к сессии атрибутов, необходимых для реализации ролевой модели разграничения доступа, собственно же авторизацией должен заниматься сам информационный источник. Взаимодействуя с сервисом аутентификации, сервер интеграции и доступа может как сам обеспечить авторизацию (для чего используются специальные декларации в поддерживаемом наборе метаданных), так и передать необходимые атрибуты информационному источнику (внеся их в предусмотренные параметры в интерфейсах адаптера) для самостоятельного решения вопросов авторизации.

Служба поиска обеспечивает формирование полнотекстовых и атрибутных запросов пользователей портала на поиск данных в информационных источниках посредством набора поисковых форм, а также форматирование и вывод результатов. Задача службы поиска – формирование запроса в терминах виртуальной (канонической) схемы портала и исполнение этого запроса посредством обращения к серверу интеграции и доступа. Запросы строятся на основе описаний информационных объектов в канонической схеме.

1.2. Адаптеры информационных источников

В существующей версии портала реализована серия современных концепций, обеспечивающих построение распределенных web-систем, объединяющих на федеративной основе различные информационные источники – поставщики контента. Среди этих решений имеется и базовая концепция формирования пользовательского интерфейса, предполагающая разделение кода получения данных и кода представления. Весь код в портале разделен на две части: одна часть решает задачи

получения/передачи собственно данных, другая – интерфейсные вопросы при взаимодействии с пользователем. Множественность информационных источников свидетельствует о сложности задачи получения/передачи данных. За счет выделения задачи обмена данными между информационными источниками и порталом в самостоятельный блок всему остальному коду портала обеспечивается независимость от информационных источников, т.е.

- независимость от способа взаимодействия с источником (от конкретного программного интерфейса адаптера внешней системы);
- независимость от форматов данных, принятых в источниках;
- независимость от способа хранения данных в источнике;
- независимость от расположения источника (т.е. одинаковые условия выполнения и в локальной, и в распределенной средах, абстрагирование от таких параметров, как адреса, URL и т.п.).

Любые данные, используемые кодом представления, поставляются сервером интеграции и доступа, который, в свою очередь, получает их, взаимодействуя с внешней системой посредством адаптера – представителя этой системы в портале. Такой подход, как уже упоминалось, свойственный многим портальным решениям, можно считать заимствованной технологией, развитой в рамках традиционных электронных библиотек [6]. Однако у портальных адаптеров есть и отличия в сравнении с традиционными адаптерами, выполняющими «аналогичную» функцию в электронных библиотеках. От традиционного адаптера требуется обеспечить только интерфейс к данным, обслуживаемым информационным источником. Таким образом, задача адаптера состоит в переводе запросов, сформулированных в терминах виртуальной (канонической) схемы репозитория (портала в данном случае), в запрос в терминах схемы источника. При этом, естественно, адаптер должен поддерживать язык запросов репозитория и обеспечивать преобразование не только схем, но и запросов на языке репозитория в запросы на языке источника, который в общем случае отличается (возможно, принципиально отличается, а то и отсутствует вовсе) от языка репозитория. В итоге задача создания адаптера может оказаться крайне сложной, сопоставимой (а то и более трудоемкой) с задачей создания собственно источника. В портале такая сложная схема не нужна. Предполагается, что можно ограничить взаимодействие портала и информационного источника конечным (не очень большим) перечнем запросов, каждому запросу поставить в соответствие ровно одну команду и реализовывать в адаптере только ее. Благодаря этому предложению адаптер портала, а также сервер интеграции и доступа, да и вся технология взаимодействия с внешними системами приобретают следующую специфику:

- функциональный интерфейс (универсальный язык запросов) хотя и существует, но очень прост, есть только определения команд;
- отсутствует, вообще говоря, схема источника; сама виртуальная схема репозитория портала представляет собой, по сути, набор описаний информационных объектов с методами доступа к ним.

1.3. Схема работы портала

Из сказанного выше вытекает следующая схема работы портала.

1. Любой код портала (в частности, при обслуживании пользовательских вызовов – код шаблона страницы сайта портала) работает с данными, используя набор данных, находящихся в экземпляре класса DataSet.

2. Для получения этого набора данных соответствующий код обращается к серверу интеграции и доступа, предавая ему запрос и параметры, необходимые для формирования соответствующей команды. При необходимости выполнения нескольких команд код может обратиться к серверу интеграции и доступа несколько раз или передать командный запрос – последовательность нескольких команд – и параметры для их выполнения. Так, например, сайт портала поддерживает собственные конфигурационные параметры, ставящие в соответствие каждому URL определенный командный запрос [3]. Другой код может непосредственно содержать последовательность вызовов.

3. Для каждого запроса сервер интеграции и доступа, пользуясь виртуальной схемой репозитория портала и переданными параметрами, формирует набор команд, исполняемых конкретными

адаптерами. При необходимости обращения к нескольким источникам выполнение команд осуществляются параллельно.

4. Код реализации адаптера выполняет команду (в простейшем случае просто перенаправляет вызов источнику, если тот сам поддерживает нужный программный интерфейс, в более сложном – сам работает с данными источника).

5. Результаты работы всех адаптеров возвращаются серверу интеграции и доступа, объединяются в один набор DataSet в соответствии с правилами объединения (например, в соответствии с системой наследования типов), описанными в схеме, и направляются осуществившей вызов исходного запроса службе.

Сервер интеграции и доступа, таким образом, обеспечивает возможность работы с множеством подключенных информационных источников, предоставляющих порталу структурированный контент, как с единым хранилищем. Источники структурированной информации, подключаемые к порталу через сервер интеграции и доступа, могут быть либо одного из предопределенных типов, либо некоторого специального типа. Для каждого предопределенного типа источника уже имеется реализация класса, на основе которой может быть легко сгенерирован соответствующий адаптер. В настоящий момент таких типов три:

- внутреннее хранилище портала (сервер приложений, обеспечивающий поддержку собственного контента в рамках сайта портала);
- ADO.NET-источники;
- SOAP-источники.

Подключение специального типа источника требует либо разработки специализированного модуля, предназначенного для генерации адаптера для подключения именного этого типа информационного источника к порталу, либо фрагмента кода (реализации функции доступа) на языке C#. В настоящий момент есть серия таких источников, из которых можно выделить архивы выдающихся ученых [7] и Интегрированную систему информационных ресурсов РАН [8,9]. Требования со стороны портала к разработке модулей для таких источников минимальны и обсуждаются далее.

2. Виртуальная схема данных

Работа сервера интеграции и доступа основана на использовании виртуальной (канонической) схемы данных. Схема включает описания структур данных, поддерживаемых источниками портала, различные метаданные, содержащие описания способов доступа к информации, предоставляемой информационными источниками, и служебную информацию, используемую подсистемами портала, которые пользуются схемой. Вся совокупность сформированных таким образом метаданных размещается в файлах XML и условно делится на две части: *репозиторий типов* и *систему реализации типов*.

2.1. Репозиторий типов

Репозиторий типов описывает множество типов, каждый из которых содержит атрибуты и определения методов, доступных посредством сервера интеграции и доступа. Для каждого типа в репозитории типов содержится следующая информация:

- атрибуты – описание имен и типов полей;
- индексы – первичный ключ и уникальные ключи;
- связи с другими типами – описание взаимосвязей объектов данного типа с другими с указанием способа получения связанных объектов;
- указатели на базовые типы – средство для построения деревьев наследования типов (включая множественное наследование) и необходимых атрибутов для управления поведением полей и методов в порядке наследования;
- набор определений методов, обеспечивающих манипулирование с объектами данного типа.

Репозиторий типов является интерфейсной частью виртуальной схемы данных портала, которая отображает общую структуру данных, абстрагированную от деталей реализации доступа и преобразования форматов. Репозиторий типов основывается на трех основных элементах – тип, связь, мультиязычное описание.

Тип представляет собой совокупность атрибутов, которые могут быть представлены в формате XML-Schema [10]. Каждый тип однозначно идентифицируется свойством «name» и, соответственно, существование двух типов с одинаковым свойством «name» невозможно.

Репозиторий типов реализует множественное наследование. Каждый тип может иметь один или несколько базовых типов: наследование означает включение в состав атрибутов данного типа всех атрибутов базовых типов. Если атрибуты базовых типов пересекаются по названиям, результирующий тип будет содержать один унаследованный атрибут с данным именем.

В качестве примера рассмотрим описание типа с именем «Person», для объектов которого предусматриваются два метода «Get» (получить экземпляры) и «Set» (модифицировать экземпляр).

```
…
<Type name="Person" entity="Person">
        <Field name="Id" entity="Id"/>
        <Field name="Name" entity="Name"/>
        <Field name="Surname" entity="Surname"/>
        <Field name="BornDate" entity="BornDate"/>
        <Field name="Address" entity="Address"/>
        <Script name="Get">
                <Parameter name="Id" entity="Id" optional="true"/>
                <Condition name="Name" entity="Name">
                        <Operation name="IN"/>
                </Condition>
                <Condition name="BornDate">
                        <Operation name="equal"/>
                        <Operation name="greater"/>
                </Condition>
                ...
        </Script>
        <Script name="Set">
                <Parameter name="Id" entity="Id" optional="false"/>
                <Parameter name="Name" entity="Name" optional="true"/>
                <Parameter name="Surname" entity=" Surname" optional="true"/>
                <Parameter name=" BornDate" entity=" BornDate " optional="true"/>
                <Parameter name=" Address" entity="Address" optional="true"/>
        </Script>
        …
</Type>
```

Для иллюстрации механизма наследования ниже приведен пример описания типа с именем «scientific», который расширяет базовый тип «Person» полем «Academic_degree»:

```
<Type name="Scientific" entity="Person">
        <Field name="Academic_degree" entity=" Academic_degree"/>
</Type>
```

Каждому типу соответствует *мультиязычное описание* (элемент «Entity»), которое используется для представления пользователю литеральных данных. Так, например, типу, который описывает личность, может соответствовать описание, содержащее выводимое имя (атрибут «DisplayName») на трех языках – русском, английском и французском, типу, описывающим адрес – мультиязычные дескрипторы.

```
<Entity name="Person">
        <DisplayName lang="ru">Персона</DisplayName>
```

```
        <DisplayName lang="en">Person</DisplayName>
        <DisplayName lang="fr">Personne</DisplayName>
</Entity>
<Entity name="Address" type="System.String">
        <DisplayName lang="ru">Адрес</DisplayName>
        <DisplayName lang="en">Address</DisplayName>
        <Description lang="ru">Местожительства</Description>
        <Description lang="en">Residence</Description>
</Entity>
```

Мультиязычные описания можно поставить в соответствие любому типу и любому атрибуту типа. Несколько типов или атрибутов могут быть соотнесены к одному описанию. Описание определяется атрибутом «entity» в представлении типа или атрибута.

Репозиторий типов поддерживает определение *связей* между типами. Связь определяется в описании типа указанием имени связанного типа, имени функции, которая может вернуть связанные объекты, и типа связи (к одному/ко многим). Ниже приведен пример описания связи типа «Person» с типом «Car» (автомобиль). Окраску этой связи естественно будет охарактеризовать дескриптором «Владелец».

```
<Entity name="Owner" type="System.String">
        <Description lang="ru">Владелец</Description>
        <Description lang="en">Owner</Description>
</Entity>
…
<Type name="Car" entity="Car">
…
</Type>
<Type name="Person" entity="Person">
…
        <Link name="Car" scriptname="Link_Person_Script" occurs="multiple" entity="Owner"/>
</Type>
```

2.2. Система реализации типов

Репозиторий типов представляет собой схему, общую для всех информационных источников, которые подключены к порталу. *Система реализации типов* предназначена для поддержки взаимодействия с информационными источниками и может быть рассмотрена как набор локальных схем подключенных источников, выраженных в терминах виртуальной схемы. Это означает, что в рамках портала реализуется подход к описанию источников *Global As View* (глобальная схема). Но при этом задача перехода от терминов глобальной схемы к терминам локальных схем источников решается не столько адаптером источника, сколько самой системой метаданных виртуальной схемы портала. В рамках этой же системы реализуется и *репозиторий источников*: для каждого информационного источника в системе реализации типов выделяется отдельная секция, которая содержит следующую информацию:

- данные о самом источнике: вид, способы взаимодействии и т.п.;
- подмножество типов репозитория типов, которые поддерживает источник;
- реализации методов этих типов.

В секции источника содержатся все необходимые данные для создания среды взаимодействия с ним. Например, для взаимодействия с реляционной базой данных в эту секцию включается информация для создания сессии (настройки источника ODBC), для web-сервиса – WSDL-описание или URL, по которому его можно получить. Кроме того, в секции источника содержится описание отображения типов и связей из репозитория типов на данный источник. В этом отображении присутствуют только те типы и связи, которые реализуются данным источником (его подсхема). При помощи отображения типов производятся операции над экземплярами типов (под экземплярами в данном контексте понимается структура данных, соответствующая описанию типа).

Для каждого метода, определение которого включено в репозиторий типов, в системе реализации типов имеется секция, задающая реализацию данного метода. Система реализации типов позволяет определять статические и динамические методы, а также модули исполнения с использованием языков программирования, поддерживающих CLR [11], включая фрагменты кода в текстовое определение метода. Ниже приводится пример описания реализации методов «Get» и «Set» для типа «Person».

```
<Type name="Person">
…
<Script name="Get">
<Body>
    <![CDATA[SELECT %fields% FROM PERSON %conditionsandparameters% %orders%]]>
</Body>
</Script>
<Script name="Set">
<Parameterspattern>
      ({0[[NativeFieldName]]:*: , }) VALUES ({0[[@[Name]]]:*: , })
</Parameterspattern>
<Body>
    <![CDATA[UPDATE PERSON SET
              <% foreach (PSSourceCondition _cond in Conditions) { If (_cond.Name != "id")
              _result.Append(_cond.NativeFieldName + "=@" + _cond.Name + ", ") %>
      WHERE Id=@Id
    ]]>
</Body>
</Script>
</Type>
```

Приведенный пример иллюстрирует возможности формирования из реализации метода команды, исполняемой адаптером источника, – запроса выборки (SELECT) и запроса модификации (UPDATE) на языке SQL. Для формирования команды применен язык программирования C#, фрагменты программного кода заключены в маркеры «<%» и «%>». Команда конструируется путем выполнения фрагментов кода, входящих в реализацию метода, и объединения их с текстовыми фрагментами в этой же секции реализации. В приведенном примере фрагмент команды UPDATE будет включен в результирующий текст команды только при выполнении условия _cond.Name != "id".

В коде, который приводится в реализации метода, могут быть использованы переменные «Conditions», «Fields», «Lang» и «User». Переменная «Conditions» представляет собой коллекцию условий. Каждое условие выражается тройкой «поле-значение-операция» и выражает критерии отбора данных. В переменной «Fields» содержится коллекция имен терминов, которые должны быть возвращены. Параметр «Lang» определяет язык, с его помощью реализуются мультиязычные свойства портала. Параметр «User» используется для интеграции с подсистемой безопасности портала. Он представляет собой объект, реализующий интерфейс IPrincipal, через который доступна информация о вызывающем пользователе [4].

В фрагментах кода, обеспечивающего формирование текста команды, можно использовать предопределенную переменную «_result» строкового типа, которая содержит уже сформированную часть команды.

Существует также еще одно средство для конструирования команд – это шаблоны подстановки %fields%, %conditions%, %parameters% и т.п. При генерации текста команды эти идентификаторы будут заменены на строки, которые формируются, соответственно, из коллекций «fields», «conditions», «parameters» с применением шаблона подстановки. Шаблон подстановки представляет собой строку, которая отражает способ преобразования массива элементов в строковый вид с использованием разделителей. Например, если коллекция «fields» содержит элементы «id» и «name», то, применив шаблон подстановки «{0[]:*:, }», получим строку «id, name». Шаблоны

подстановки могут быть определены для источника данных, для реализации типа и для конкретного метода. Они описываются в секциях «Fieldspattern», «Conditionspattern», «Parameterspattern».

Сгенерированная команда должна быть передана адаптеру источника для выполнения и получения результирующего набора данных. Для этих целей служит секция источника, которая называется «Exec». В этой секции определяется код для манипулирования с данными, а именно код, который обрабатывает созданную команду и формирует результирующий набор данных. В качестве параметров передается сама команда и описанные выше параметры «fields», «conditions», «lang», «user».

Ниже приведен пример определения обработки команды в секции «Exec» для базы данных SQL Server.

```
<Exec>
_result = Utils.ExecSqlSource("workstation id=localhost;packet size=4096;integrated security=SSPI;data
source=localhost;persist security info=False;initial catalog=CATALOG", Script, Conditions, Lang, User);
</Exec>
```

В приведенных примерах, хотя это и не указывалось явно, в качестве внешнего источника фигурировала база данных SQL. Покажем, каким образом тип «Person» может быть реализован для источников данных, не поддерживающих язык запросов SQL. Прежде всего к таким источникам нужно отнести web-сервисы [12]. Приведем пример реализации данного типа для web-сервиса, обеспечивающего две функции: «GetPerson» (выборка данных о персонах) и «SetPerson» (модификациях данных о персоне). Далее приведено соответствующее описание, размещенное в системе реализации типов.

```
<Type name="Person">
<Parameterspattern>
      {0<[NativeFieldName]>[Value]</[NativeFieldName]>:*: }
</Parameterspattern >
<Script name="Get">
<Body>
    <![CDATA[
        <?xml version="1.0" encoding="utf-8"?>
        <soap:Envelope
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns:xsd=http://www.w3.org/2001/XMLSchema
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Body>
                < GetPerson xmlns="http://tempuri.org/">
                        %parameters%
                </GetPerson>
        </soap:Body>
        </soap:Envelope>
    ]]>
</Body>
</Script>
<Script name="Set">
<Body>
    <![CDATA[
        <?xml version="1.0" encoding="utf-8"?>
        <soap:Envelope
        xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
        xmlns:xsd=http://www.w3.org/2001/XMLSchema
        xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
        <soap:Body>
                < SetPerson xmlns="http://tempuri.org/">
                        %parameters%
                </SetPerson>
```

```
        </soap:Body>
        </soap:Envelope>
    ]]>
</Body>
</Script>
</Type>
```

Реализация в данном случае обеспечивает формирование текста SOAP-запроса для каждого метода, а потом этот текст передается функции «Exec» из секции источника. Реализация «Exec» содержит библиотечную функцию «ExecSOAPSource», которая направляет полученный SOAP-запрос web-сервису, получает результат и преобразует его в нужный вид. Секция «Exec» в данном случае выглядит следующим образом:

```
<Exec>
      _result = Utils.ExecSOAPSource("http://www.ras.ru/testsvc/", Script, Conditions, Lang, User);
</Exec>
```

Реализация типа «Person» для web-сервиса построена без применения выражений на C#: используется только шаблон подстановки %parameters%, общий для методов «Get» и «Set». Это обусловлено спецификой построения текста SOAP-запроса: он должен содержать XML-теги с именами и значениями передаваемых параметров. Для этого выразительных способностей шаблона подстановки вполне достаточно, причем он обобщен для всех реализуемых функций.

Еще одним примером может служить реализация взаимодействия с XML-хранилищем (например, с XML-файлом). В этом случае серверу интеграции и доступа необходимо построить выражение на языке XPath [13]. Поскольку XPath не предполагает модификацию данных, возможно реализовать только метод «Get».

```
<Type name="Person">
<Script name="Get">
<Parameterspattern>
{0@[NativeFieldName]=“[Value]”:*: and }
</Parameterspattern>
<Body>
    <![CDATA[//Persons/Person[%parameters%]]]>
</Body>
</Script>
```

Для получения данных из XML-файла функция «Exec» секции источника использует XML-парсер, которому передается сгенерированное XPath-выражение:

```
<Exec>
      XmlDocument _doc = new XmlDocument();
      _doc.Load(“c:\work\xmlstorage.xml”);
      XmlNodeSet _nodes = _doc.SelectNodes(Script);
      _result = Utils.GenerateDataSetFromXmlNodeSet(_nodes);
</Exec>
```

Результат выборки преобразуется в набор DataSet при помощи функции «GenerateDataSetFromXmlNodeSet».

Таким образом, предложенные механизмы позволяют реализовывать процедуры манипулирования практически для любых источников данных, даже не поддерживающих какой-либо язык запросов.

Завершая обсуждение виртуальной схемы данных портала, отметим, что создание и поддержка этой схемы – задача администратора портала. Именно он должен решать, как разные источники отображаются на схему данных портала. Простейшим примером является ситуация, когда схема просто содержит для каждого типа каждого информационного источника отдельный тип, содержащий набор методов с собственным описанием возвращаемых результатов, т.е. никакие инстру-

менты интеграции попросту не задействуются. Другой, «крайний» пример, когда создается некоторая всеобъемлющая схема данных, полностью описывающая предметную область, со связями и наследованием, а затем реализуется набор адаптеров, поддерживающих подмножество этих типов, т.е. выполняющих запросы, осуществляя преобразование структуры возвращаемых данных к схеме репозитория портала. Естественно, что деятельность администратора портала в этой части поддержана необходимыми инструментальными средствами.

3. Реализация сервера интеграции

Функционально сервер интеграции и доступа реализуется *генератором схемы, адаптерами источников* и *процессором запросов.*

3.1. Генератор схемы и адаптеры источников

Генератор схемы обеспечивает загрузку схемы (файлов XML), добавление/модификацию определений в репозитории типов и систему реализации типов и выполняет генерацию адаптеров источников. По сути, описание реализации набора типов для источника и является декларативным определением адаптера для этого источника. Генератор схемы предназначен для компиляции набора сборок (модулей) в памяти, реализующих функционал адаптера в виде классов .NET. После компиляции каждая сборка загружается в память и создаются экземпляры сгенерированных классов, представляющие собой оболочки для генерации и исполнения запросов к данному источнику. Компиляция и загрузка выполняются при начальной инициализации схемы, а также при добавлении описаний реализации типов в схему. Соответственно, при исключении источника из схемы происходит выгрузка соответствующих модулей. Таким образом достигается динамическое манипулирование составом подключенных источников без остановки работы всей службы.

При подключении нового источника к существующей схеме администратору необходимо сделать выбор: либо добавлять к существующей виртуальной схеме новые типы и процедуры, либо реализовывать существующие типы для вновь подключаемого информационного источника, описывая, при необходимости, правила преобразования параметров и результатов. Решение по каждому вновь подключаемому источнику является индивидуальным и в значительной степени определяется семантикой подключаемого ресурса. В любом случае, внесенные в схему изменения должны поддерживаться указанной в качестве нового адаптера сборкой .Net.

Для нового информационного источника генератором схемы будет создан новый адаптер. Существуют средства для создания трех типовых адаптеров, как то: внутреннее хранилище портала, адаптер для доступа к ADO.Net источникам (на основе реляционных СУБД) и адаптер для доступа к SOAP источникам (web-сервисы). При помощи этих средств создание нового адаптера становится тривиальной задачей. Внутреннее хранилище портала является развитым сервером приложений, обеспечивающим работу с собственными типами портального контента [3]. Функционал для создания адаптера, обеспечивающего доступ к ADO.Net источникам, позволяет в качестве метода определить вызов хранимой процедуры или выполнение последовательности операторов языка SQL (SQL Batch) для определенного реляционного источника. Функционал для создания адаптера для доступа к SOAP источникам позволяет вызывать функции указанного web-сервиса, осуществляя при этом преобразование результатов вызова функции web-сервиса к массиву объектов DataTable по заданным правилам.

Существует также возможность использования других сборок, в которых могут быть реализованы функции доступа к данным, формирования команд и др. Для этих целей используется секция источника «Assembly», которая содержит путь к сборке, ее полное имя и другие необходимые данные. Таким образом, автоматическая генерация адаптеров не является единственным средством подключения источника к порталу: для «сложных» источников можно создавать адаптеры и «вручную».

3.2. Процессор запросов

Процессор запросов осуществляет взаимодействие со сгенерированными адаптерами информационных источников с целью выполнения запроса и формирования общего результата операции. Через точку входа, которой является функция исполнения «Exec» (перегруженная), процессор запросов получает запрос в терминах схемы. Этот запрос содержит набор источников, которые должны участвовать в операции, имя типа, который должен быть задействован, метод типа, который должен быть выполнен, набор параметров вызова, набор полей, которые должны быть возвращены в случае, если метод возвращает таблицу, а также набор условий (если необходимо). Набор источников может быть не задан. В этом случае используются все источники, которые реализуют данный тип.

Хотя в параметрах вызова функции исполнения передается тип, с объектом которого производится операция, реально количество типов может быть больше – операция может охватывать как только определенный в параметрах тип, так и все унаследованные от него типы, поэтому сначала процессор определяет полный набор типов, задействованных в операции.

Далее процессор запросов определяет набор источников, участвующих в операции. Это или переданный в параметрах набор источников, или все источники, реализующие определенный выше набор типов. Для каждого источника процессор получает экземпляр класса, соответствующий типу. Напомним, что этот экземпляр образуется сгенерированным ранее адаптером. Далее, пользуясь этим экземпляром класса, процессор запросов генерирует команду для каждого источника на его языке и выполняет ее, вызывая соответствующий метод соответствующего класса (по сути, метод адаптера).

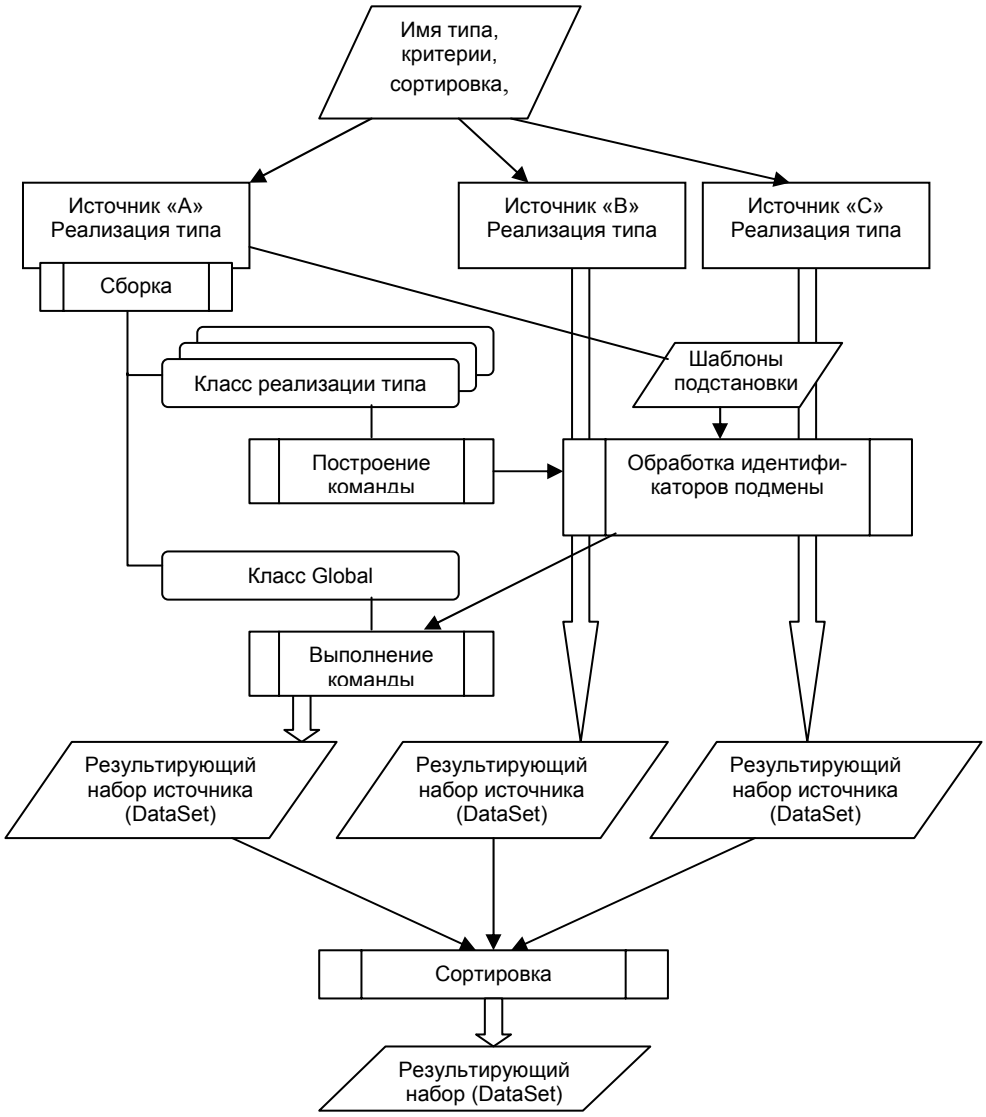
Описанная выше функциональность реализуется с помощью динамического построения и применения сборок с использованием технологии генерации кодов CodeDom и позднего связывания Reflection.

Метаданные, используемые процессором запросов, содержат фрагменты кода, определяющие функциональность при взаимодействии с информационными источниками. Эти фрагменты сосредоточены в двух местах для каждого источника: в описании реализации типа в системе реализации типов и в секции «Exec». Для взаимодействия с информационным источником процессор запросов предварительно компилирует этот код в исполняемый модуль – сборку (assembly) в терминах .Net, а при выполнении запроса вызывает методы объектов сборки по заданным правилам.

Рассмотрим принципы генерации сборок. Для каждого источника генерируется своя сборка, содержащая определения классов в соответствии с типами, реализуемыми источником, и предопределенный класс Global. Класс Global реализует единственный метод «Execute», который включает код, содержащийся в секции «Exec» источника. Каждый класс, соответствующий реализуемому типу, содержит методы формирования команд. Метод формирует команду из фрагментов кода и фрагментов текста, описанных в секции «Script» для соответствующего метода реализуемого типа. Возвращаемое значение такого метода – строка, которая является сгенерированной командой. Приведем пример кода метода «Set», сгенерированного по описанию типа «Person»:

```
public class person {
    public virtual string Set(PortalCore.PSFieldsEnumeration Fields,
                                    PortalCore.PSConditions Conditions, string Lang, string User) {
        System.Text.StringBuilder _result = new System.Text.StringBuilder();
        _result.Append("UPDATE PERSON SET");
        foreach (PSSourceCondition _cond in Conditions)
                if ((_cond.Name != “id”) {
                _result.Append(_cond.NativeFieldName + “=@” + _cond.Name + “, ”)
      }
      return _result.ToString();
  }
}
```

После построения сборок становится возможным обслуживание запросов к данным источников. Общая последовательность операций представлена на схеме.



Последовательность операций с данными

Процессор запросов в качестве исходных параметров принимает имя типа, который участвует в операции, набор условий/параметров, условия сортировки, набор возвращаемых полей, язык, на котором должны быть выданы результаты, и учетные данные вызывающего пользователя. Далее определяется список информационных источников, которые реализуют данный тип с помощью системы реализации типов виртуальной схемы. Для каждого типа выполняется операция построения и исполнения команды, результатом которой является набор данных DataSet (для операций выборки). Результирующие наборы данных по всем участвующим в запросе информационным источникам объединяются в один и производится сортировка, после чего этот результирующий набор данных выдается инициировавшей запрос службе портала.

Рассмотрим подробнее процесс построения и выполнения команды для источника. Как упоминалось выше, каждому типу, реализуемому информационным источником, соответствует сгенерированный класс. Процессор запросов в первую очередь создает экземпляр класса, соответствующий типу, и вызывает для построения команды метод, который соответствует операции поиска.

Этот метод возвращает текст сконструированной команды. Текст команды на данном этапе может содержать идентификаторы подстановки (%fields% и %conditions%). Используя шаблоны подстановки для коллекций «Fields» и «Conditions», процессор запросов строит их строковые отображения и осуществляет подмену идентификаторов этими строковыми отображениями. На этом этапе формирование команды завершено.

После формирования команды процессор запросов, используя экземпляр класса Global сборки адаптера источника, вызывает метод «Execute», в который помимо прочих параметров передается сгенерированная команда. Метод «Execute» обрабатывает команду и возвращает результат в виде набора данных.

Заключение

Описанная технология взаимодействия с информационными источниками в рамках Информационного web-портала реализована в полном объеме в существующем решении, использованном, в том числе, в портале РАН www.ras.ru. Эту технологию уместно охарактеризовать как технологию *функциональной интеграции*. Любой подключающийся к порталу источник может на основе сформулированных требований проделать следующие шаги:

1. Описать функциональный интерфейс к своим данным с учетом накладываемых сервером интеграции и доступа ограничений.

2. Решить, каким образом и будут ли данные источника интегрироваться с данными других информационных источников. Если нет, то функциональный интерфейс адаптера просто добавляется к виртуальной схеме. Если да, то для части типов (или для всех) уже есть аналоги в схеме, поэтому эти типы должны быть реализованы, в т.ч. должны быть описаны правила объединения результатов в блоке реализации каждого метода.

3. Подготовить определение своего адаптера, описав его в терминах системы реализации типов.

4. Подключить источник к серверу интеграции и доступа, используя имеющийся административный интерфейс.

В итоге внешний информационный источник будет обеспечен следующим сервисом: данные и операции с информацией источника (в рамках функционально интегрированной части, т.е. в соответствии с перечнем реализованных в адаптере методов) будут доступны в едином стиле и общей структуре представления Информационного web-портала.

Литература

1. Глеб Галкин. Такие разные корпоративные порталы // «Сетевой журнал», №5, 2002 г. http://www.setevoi.ru/cgi-bin/text.pl/magazines/2002/5/54.
2. Босов А.В., Полухин А.Н. Об организации доступа к данным в Информационном web-портале // Труды 8-ой Всероссийской научной конференции «Электронные библиотеки: перспективные методы и технологии, электронные коллекции» - RCDL'2006, Суздаль, Россия, 2006. С.301-308.
3. Босов А.В., Иванов А.В. О реализации системы управления содержанием информационного Web-портала // Информационные технологии и вычислительные системы. №4. – М., 2004. С.85-103.
4. Босов А.В., Полухин А.Н. О реализации сервиса аутентификации web-портала // Информационные технологии и вычислительные системы. – М., 2005. №3. С.50-60.
5. Босов А.В., Иванов А.В. Программная инфраструктура Информационного web-портала РАН // Программирование, №4, 2007.
6. Информационно-интерактивный портал «Российские электронные библиотеки» // http://www.elbib.ru
7. Босов А.В., Чавтараев Р.Б. Научное наследие: личные архивы выдающихся ученых в портале РАН // Системы и средства информатики. Специальный вып. «Научно-методологические проблемы информатики» – М.: Наука, 2006. С.461-475.
8. Бездушный А.А., Бездушный А.Н., Нестеренко А.К., Серебряков В.А., Сысоев Т.М. Предложения по наборам метаданных для научных информационных ресурсов ЕНИП РАН // Электронный журнал, посвященный созданию и использованию электронных библиотек, том 7, выпуск 5. Москва: Институт развития информационного общества - 2004. http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2004/part5/bbnss.

9. Бездушный А.А., Нестеренко А.К., Сысоев Т.М., Бездушный А.Н., Серебряков В.А. Возможности технологий ИСИР в поддержке Единого Научного Информационного Пространства РАН // Электронный журнал, посвященный созданию и использованию электронных библиотек, том 6, выпуск 4. Москва: Институт развития информационного общества - 2004. http://www.elbib.ru/index.phtml?page=elbib/rus/journal/2004/part6/bnsbs.
10. XML Schema Part 2: DataTypes, Primitive DataTypes // http://www.w3.org/TR/xmlschema-2/.
11. Рихтер Дж. Программирование на платформе Microsoft .NET Framework. Мастер класс: Пер. с англ. – 3-е изд. – М.: Русская редакция, 2005; – СПб.: Питер, 2005.
12. Web Services Activity // http://www.w3.org/2002/ws/#drafts.
13. XML Path Language (XPath) 2.0 // http://www.w3.org/TR/xpath20/.

Босов Алексей Вячеславович. Родился в 1969 году. Окончил Московский государственный авиационный институт в 1993 году. Кандидат физико-математических наук, доцент, автор более 50 научных работ. Специалист в области прикладной математики и информатики. Область научных интересов – информационные технологии, аналитические и обучающие системы, Интернет-технологии и порталы. Заведующий сектором Института проблем информатики РАН.

Чавтараев Рустам Баширович. Родился в 1972 году. Окончил Московский институт радиоэлектроники и автоматики в 1995 году. Автор 12 научных работ. Специалист в области автоматизированных систем обработки информации. Область научных интересов – информационные технологии, Интернет-технологии и порталы. Старший научный сотрудник Института проблем информатики РАН.