УДК 004.65.3

В.В. Росинский

Кроссплатформенность и интероперабельность в корпоративных информационных системах на основе *Web*-сервисов

Проведено исследование технологии *Web Services*. Определены ее преимущества для разработчиков корпоративных информационных систем с распределенной архитектурой. Предложена модель использования *Web*–технологий при проектировании приложений с трехуровневой архитектурой в среде *Delphi*.

The research of the Web Services technology is conducted, its advantages for the developers of corporate information systems with the distributed architecture are defined. The model of the use of Web-technologies is suggested at an applications programming by a three-level architecture in the Delphi environment

Проведено дослідження технології *Web Services*. Визначено її переваги для розробників корпоративних інформаційних систем з розподіленою архітектурою. Запропоновано модель використання *Web*-технологій при проектуванні програм з трирівневою архітектурою в середовищі *Delphi*.

Введение. Особенностью архитектуры современных корпоративных информационных систем (КИС) есть ее распределенный (многоуровневый) характер. Такие КИС обеспечивают доступность общих корпоративных правил и служб управления содержанием для широкого спектра клиентских приложений и корпоративных систем, не требуя при этом, чтобы службы адаптировались отдельно для каждого типа клиентского приложения. Это означает, что выполнение всей бизнес-логики и процедур будет происходить на серверном уровне, что делает пользовательские приложения унифицированными и тонкими, и основная их задача – отобразить содержание из корпоративной информационной среды на предметном уровне пользователя. С другой стороны, параллельный характер распределенной архитектуры дает возможность увеличения емкости хранилищ данных, сети и серверов для повышения производительности и пропускной способности, так как в сети может появляться все больше клиентских приложений и новых информационных систем.

Постановка задачи

С развитием КИС, обладающих сложной, зачастую гетерогенной структурой, одна из ключевых задач – обеспечение необходимого уровня интеграции корпоративных приложений. Исследование современных подходов к интеграции в системах с распределенной архитектурой пришло к тому, что все они основаны на связывании подсистем через промежуточный программный слой. Наиболее существенные проблемы, возникающие при этом – обеспечение кроссплатформенности и интероперабельности, т.е. независимости программного обеспечения от операционной среды, в которой оно должно функционировать. Программное обеспечение должно сохранять свою работоспособность при изменении количества и типов процессоров (или ядер) вычислительной системы. При этом доработка программ должна быть минимальной и не затрагивающей всю иерархию программных средств.

Современные варианты построения распределенной архитектуры

Распределенная архитектура включает в себя сервер, приложения-клиенты, сервер приложений. Сервер приложений – промежуточный уровень, обеспечивающий организацию взаимодействия клиентов и сервера, например, выполнение соединения с сервером, разграничение доступа к данным и реализацию бизнес-правил. Сервер приложений реализует работу с клиентами, расположенными на различных платформах, т.е. функционирующими на компьютерах различных типов и под управлением различных операционных систем (ОС). Основные достоинства распределенной архитектуры клиент-сервера:

• снижение нагрузки на сервер;

• упрощение клиентских приложений;

• единое поведение всех клиентов;

• упрощение настройки клиентов [1].

Поскольку в КИС с распределенной архитектурой клиент и сервер приложений в общем случае располагается на разных машинах, связь клиента с сервером приложений реализуется с помощью той или иной технологии удаленного доступа:

• при помощи технологий *COM* или *CORBA*, при этом на рабочей станции запускается объект, теоретически обладающий доступом ко всем ресурсам компьютера;

• сервер *MTS* (сервер транзакций *Microsoft*) – дополнение к технологии *COM*, предназначенный для управления транзакциями;

• сокеты *TCP*/*IP* (транспортный протокол/протокол Интернета) – используется для соединения компьютеров в различных сетях, в том числе в Интернете;

• *SOAP* (простой протокол доступа к объектам) – служит универсальным средством обеспечения взаимодействия с клиентами и серверами *Web*-сервисов на основе кодирования *XML* и передачи данных по протоколу *HTTP* [2].

При выборе способа взаимодействия необходимо учитывать, что архитектура системы может измениться, и тогда возникает потребность в ее перестройке или объединении с другой системой. Интероперабельность должна достигаться за счет использования стандартных и открытых протоколов [3].

С другой стороны, интеграция в рамках КИС, как правило, осуществляется с помощью описанных ниже адаптеров.

JDBC адаптер

JDBC адаптер позволяет организовывать передачу данных между базами данных. Адаптер преобразует содержимое базы в формат *XML* и обратно. Использование адаптера *JDBC* не требует от интегрируемой системы наличия специальных сервисов, что существенно упрощает разработку и настройку модуля интеграции. К недостаткам *JDBC* адаптера можно отнести:

• замедление обработки запросов на изменение или добавление, содержащих несколько строк данных, так как адаптер в таком случае передает каждую строчку в отдельном сообщении;

• прямой доступ в базу данных может противоречить заложенной в систему логике;

• с помощью *JDBC* адаптера возможно только построение сценариев интеграции, в которых данные передаются периодически согласно некоторому регламенту.

Файловый адаптер

Файловый адаптер (*FILE*/ *FTP*) – адаптер для связи приложений на основе файлового обмена данными через *FTP* (*File Transfer Protocol*). Данный адаптер для своей работы не требует от интегрируемой системы наличия специальных сервисов, что особенно важно при работе с устаревшими, унаследованными системами. Недостатки файлового адаптера:

• при использовании файлового адаптера в сценарии интеграции появляется задержка – интервал между просмотром каталога на наличие новых файлов;

• отсутствует возможность подписи/проверки данных.

SOAP адаптер

SOAP адаптер позволяет организовать взаимодействие между удаленным клиентом и *Web*-сервисом поверх транспортного протокола *HTTP*. В адаптере *SOAP* поддерживается механизм *SSL* (*Secure Socket Layer*), что позволяет передавать данные по защищенному соединению.

К основным достоинствам адаптера *SOAP* относятся:

• *SOAP* адаптер поддерживает механизм *WSDL*, организующий автоматическую синхронизацию передаваемых структур данных.

• Передача сообщений, содержащих несколько строк данных, проводится быстрее, чем при применении *JDBC* адаптера, так как все данные передаются в одном *SOAP* (*XML*) сообщении.

• Передача данных осуществляется через *HTTP* порт, который обычно открыт в брандмауэрах.

• Соответствие принципам трехуровневого подхода к построению информационных систем, когда исключается прямой доступ в базу данных для интегрируемых приложений. Под-

держка целостности данных осуществляется механизмами самой системы, что повышает уровень безопасности и надежности работы сценария интеграции.

• Возможность построения сценариев интеграции, при которых передача данных инициируется интегрируемой системой.

• Использование стандартных средств *XML* для организации и группировки передаваемых структур данных [4].

Проблемы разработки КИС на основе распределенной архитектуры

С момента появления технологий, описанных в вышеперечисленных методах, повысилась производительность процессоров, выросли объемы и быстродействие накопителей информации, а также доля оптоволоконных каналов связи, позволяющих передавать огромные массивы данных с высокой скоростью. В результате появляются новые технологии, способные сделать КИС более открытыми, т.е. обеспечивающие более высокую степень интероперабельности [5]. Однако сегодня, с учетом сложной структуры, не существует четких подходов решения проблемы интеграции в КИС с распределительной архитектурой, основанных на использовании *Web*-технологий. Кроме того, в литературе отсутствуют примеры использования и модели практического применения *Web*-технологий для интеграции в КИС с распределенной архитектурой.

Цель статьи – детально ознакомить с технологией *Web Services*, в частности с преимуществами, которые она может дать разработчикам КИС с распределенной архитектурой, и предложить модель использования данной технологии при проектировании КИС с трехуровневой архитектурой в среде *Delphi* используя удаленные модули данных и готовые компоненты, входящие в ее состав.

Распределенная архитектура на основе *Web*-сервисов

Web Services – новая технология для развертывания распределенных вычислительных систем. Основная причина ее появления – неспособность существующих технологий, таких как объектные системы типа *COM* семейства *Microsoft* и стандарты *OMG CORBA*, в полной мере обеспечить совместимость (интероперабельность) различных программных продуктов для неоднородных распределенных систем. *Web Services* представляет собой набор услуг в виде программных приложений, идентифицированного сетевым адресом *URI* (*Uniform Ressourse Identifier*), интерфейсы и связывания (*binding*) которого определяются *XML*-средствами. Основу данной технологии составляют:

• простейшие коммуникационные Интернет-протоколы *HTTP* и/или *SMTP*;

• протоколы *SOAP* (*Simple Object Access Protocol*) для управления сообщениями в универсальном *XML*-формате;

• язык *WSDL* (*Web Services Definition Language*) описания интерфейса взаимодействия компонент распределенной системы [6].

Web-сервисы обеспечивают прямые взаимодействия через Интернет с другими агентами программного обеспечения, используя сообщения, основанные также на *XML*-формате. Данное определение *Web*-сервисов не предполагает использование *SOAP* в качестве формата или модели обработки сообщений. И при этом оно не предполагает также использование *WSDL* как языка описаний обслуживания. Однако предполагается, что более высокие уровни стека протокола *Web*-сервисов должны строиться на основе *SOAP* и *WSDL*.

Основным достижением технологии *Web*-сервисов есть совместимость всех их реализаций, независящая от поставщиков (провайдеров) вычислительных услуг и производящих их технологий. Основа этой совместимости – последовательное применение на всех уровнях предоставления услуг *Web*-сервисов стандартов *XML*-технологии. Так например, формат *SOAP*-сообщений – основной единицы передачи данных – представлен в виде *XML*-документа; описание интерфейса вычислительного сервиса в *WSDL* также представляется в *XML*-формате. *SOAP* представляет достаточно простой, основанный на *XML*-механизме, способ создания структурированных пакетов данных для обменов между сетевыми приложениями. *SOAP* содержит четыре основные компоненты:

• конверт (*envelope*), определяющий рамочную структуру сообщения в формате *XML*;

• набор правил для представления типов данных;

• соглашение о представлении вызова удаленных процедур (в режиме *RPC*);

• правила совместного выполнения протоколов *SOAP* и *HTTP*. *SOAP* может использовать также комбинацию различных сетевых протоколов, таких как *HTTP*, *SMTP*, *FTP*, *RMI*/*IIOP* [7].

Алгоритм создания КИС на основе протокоа *SOAP*

SOAP – это кросс-платформенная, кросс-языковая технология запуска объектов. Основным условием при программировании *SOAP* является то, что сервер не должен сохранять свои предыдущие состояния, т.е. результат выполнения запроса не должен зависеть от предыдущих команд, полученных сервером. Это означает, что все параметры сессии должны храниться на клиенте и передаваться серверу в составе запроса (если необходимо). Этим обеспечивается высокая устойчивость и масштабируемость системы, хотя ряд других преимуществ обычной двухзвенной архитектуры становится недоступным:

• нельзя явно управлять транзакциями с клиента (этим занимается сервер);

• нельзя заблокировать запись на время редактирования;

• нельзя одной командой передать параметры, а другой – считать результат, все должно происходить в рамках одной команды;

• нельзя работать с классической связкой «мастер–деталь», однако нужно учесть, что *TClientDataset* предоставляет для этого средство «вложенные таблицы» (*nested datasets*);

• нельзя использовать свойство *ClientDataSet.PacketRecords* > 0, так как сервер не хранит данные, которые были переданы на клиент, подобную функциональность приходится реализовывать при помощи дополнительных параметров запроса [8].

Алгоритм реализации технологии *SOAP* в среде программирования *Delphi* следующий:

Ш а г 1. После запуска *Delphi* необходимо выбрать в меню *File* | *New* | *Other* …, далее следует перейти на вкладку ***Web Services*** репозитария обьектов (рис. 1).

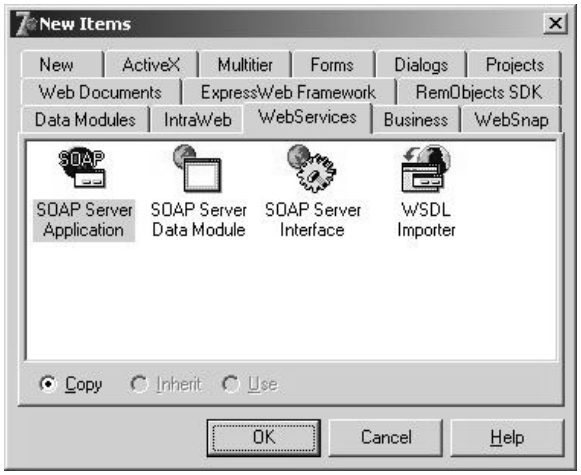

Рис. 1. *Delphi* 7. Репозитарий обьектов

Ш а г 2. Для создания приложения на основе технологии *DataSnap*, которое будет функционировать как кроссплатформеный *Web*-сервис необходимо выбрать ***SOAP Server Application***. После этого запустится мастер, в котором будут предоставлены следующие варианты:

• *ISAPI*/*NSAPI Dinamic Link Libarry* – подключаемая библиотека для серверов *IIS*/ *Netscape*, каждый запрос передается как структура и обрабатывается отдельным тредом;

• *CGI Stand-alone Executable* – консольное приложение, получает запрос на стандартный вход, возвращает ответ на стандартный выход, каждый запрос обрабатывается отдельным экземпляром приложения;

• *Win-CGI Stand-alone Executable* – приложение *Windows*, обмен данными происходит через *INI*-файл (не рекомендуется к использованию, как устаревшее);

• *Apache Shared Module* (*DLL*) – подключаемая библиотека для сервера *Apache*, каждый запрос передается как структура и обрабатывается отдельным тредом;

• *WebAppDebugger Executable* – подключаемая библиотека для отладочного сервера, поставляемого в составе *Delphi*, поскольку *WebAppDebugger* также является *COM* сервером, необходимо указать (произвольное) *CoClass Name* для *COM* объекта, с помощью которого будет вызываться ваш веб-модуль.

Поэтому следует выбрать ***CGI Stand-alone Executable***, как наиболее простой для отладки формат (рис. 2), потом приложение можно будет легко преобразовать в любой другой. Это возможно реализовать используя подход, при котором вся логика приложения будет сосредоточена в написанных модулях. В дальнейшем, если будет необходимость, создать новое приложение другого типа, к нему нужно просто подключить готовые модули. Для подтверждения выбора нужно нажать «*OK*». После чего появиться всплывающие диалоговое окно (рис. 3), в котором будет предложено создать интерфейс для модуля *SOAP*. Ввиду того, что в данный момент не стоит задача создания *Web*-риложения, следует выбрать «*No*».

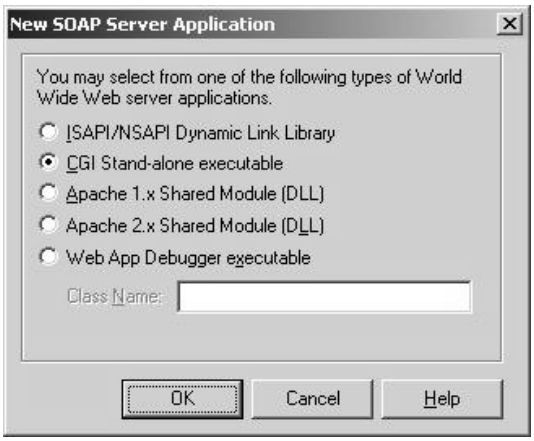

Рис. 2. Создание нового серверного *SOAP*-приложения

Рис. 3. Диалоговое окно

Ш а г 3. После этого будет сгенерировано новое приложение, содержащее *WebModule* с тремя компонентами:

• *THTTPSoapDispatcher* – получает входящие *SOAP* пакеты и передает их компоненту, определенному его *Dispatcher property* (обычно *THTTPSoapPascalInvoker*);

• *THTTPSoapPascalInvoker* – получает входящий *SOAP* запрос, находит в *Invocation Registry* вызываемый метод, выполняет (*invokes*) его, формирует ответ и передает его обратно *THTTPSoapDispatcher*;

• *TWSDLHTMLPublish* – формирует *WSDL* (*Web Services Description Language*), описание данных и интерфейсов, поддерживаемых модулем (рис. 4).

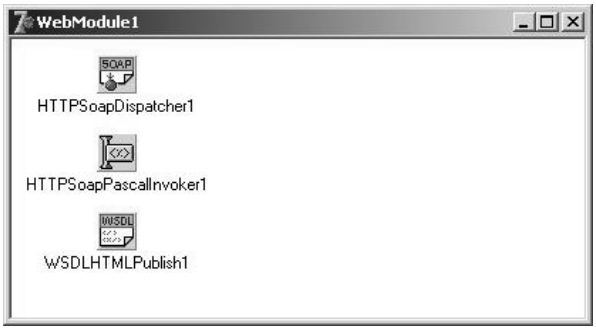

Рис. 4. *SOAP Web*-модуль с компонентами

Ш а г 4. Далее следует сохранить созданное приложение, оно будет основой сервера.

Ш а г 5. *Web*-модуль *SOAP* следует сохранит в файле *SWebMod.pas* и весь проект в файле *D7DB2CGI.dpr*.

Ш а г 6. Для соединения с базой данных в проект необходимо добавить модуль данных *SOAP*. Для этого нужно использовать второй значок на вкладке *WebServices* репозитария обьектов. В конструкторе модуля данных *SOAP* необходимо указать имя нового модуля данных – *D7DB2SAMPLE* (рис. 5).

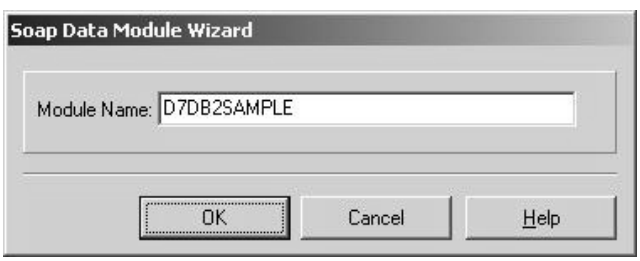

Рис. 5. Мастер нового модуля данных *SOAP*

Ш а г 7. Модуль данных *SOAP* сохраняется в файле *SDataMod.pas*. Для того чтобы приложение функционировало как *DataSnap SOAP*- сервер, необходимо использовать компоненты доступа к данным *dbExpress*^{TM}. На стороне сервера следует добавить компонент *TdataSetProvider*. На стороне клиента *DataSnap SOAP* используется *TSOAPConnection and TclientDataSet.*

Ш а г 8. Далее необходимо снова воспользоваться компонентами *dbExpress*. Сначала в

модуль данных *SOAP* добавляется компонент *TSQLConnection*. Значение свойства *ConnectionName* этого компонента нужно сменить на *DB2Connection*, затем необходимо проверить параметры соединения (есть ли у свойств *User_Name* и Пароль значения), и наконец, необходимо установить свойство *LoginPrompt* в значение ***False***. Если при этом свойство *Active* установиться на ***True*** без проблем, то это значит, что можно установить соединение к Базе данных *DB2 SAMPLE*.

Ш а г 9. Следующим шагом будет добавление в модуль данных *SOAP* компонента *TSQLDataSet* и трех компонент *TSQLTable*, по одному для каждой из трех подробных таблиц базы данных *DB2*, которые будут использоваться в этом многоуровневом приложении (*EMP_ACT*, *EMP_PHOTO* и *EMP_RESUME*).

Ш а г 10. В компоненте *TSQLDataSet* с именем *SQLdsEMP* следует установить следующие значение его свойств: свойства *Connection – SQLConnection1*, свойства *CommandType – ctTable*, а свойства *CommandText – EMPLOYEE*. Далее в модуль данных *SOAP* нужно поместить компонент *TDataSetProvider*, находящийся на вкладке *Data Access*. Затем следует установить следующие значения его свойств: свойства *Name – dspEMPLOYEE*, а свойства *DataSet – SQLdsEMP*.

Ш а г 11. Также необходимо убедиться, что для каждой из таблиц *EMP_ACT*, *EMP_PHOTO* и *EMP_RESUME* значение свойства *SQLConnection* такое же, как и свойство *TableName*.

Ш а г 12. Для построения соотношением *один–ко–многим* необходим компонент *TdataSource* (*dsEMP*), указывающий на *SQLsdEMP*. Теперь все три компонента *TSQLTable* должны указать в своих свойствах *MasterSource* на компонент *DataSource*. Далее в свойстве *MasterFields* таблиц *SQLTables* необходимо указать поля *EMPNO*, чтобы определить соотношением *один–ко–многим* снова.

Теперь модуль данных *SOAP* должен выглядеть так, как на рис. 6.

Следующим шагом будет развертывание серверного приложения на компьютере-сервере. После сохранения и компиляции проекта появится файл *D7DB2CGI.exe*, который необходимо разместить в директории скриптов *Web*-сервера или в директории *cgi-bin*. Очевидно, что после того, как приложение будет размещено в этой директории, все еще будет требоваться, чтобы оно было способно получать доступ к базе данных, поэтому придется настроить параметры соединения *SQLConnection* для того, чтобы соединиться с нужной базой данных.

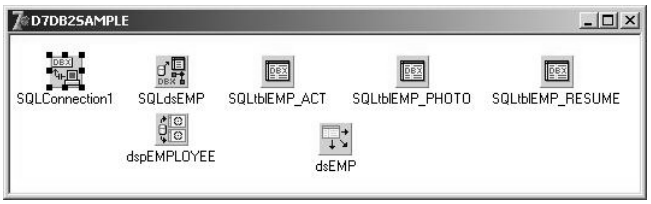

Рис. 6. Модуль данных *SOAP* с компонентами *dbExpress*

Если в адресной строке браузера ввести путь к директории, где находиться файл *D7DB2CGI.exe*: *http*:///*localhost*/*cgi-bin*/*D7DB2CGI.exe*, то в окне браузера отобразится информация о доступных сервисах на *SOAP*-сервере (рис. 7).

Стоит отметить, что *Web*-сервис предоставляет не менее четырех интерфейсов. Первые три указывают на тот же сервис, а именно на *DataSnap Web*-модуль *SOAP*, который предоставляет информацию о себе внешнему миру, используя интерфейсы *IAppServer*, *IappServerSOAP*, и *ID7DB2SAMPLE*. Если в конце адресной строки (*URL*) добавить «/*WSDL*», то получим список реализуемых *Web*-сервисом интерфейсов (рис. 8). Это формальная возможность *SOAP*-сервера, заключающаяся в предоставлении информации о своих возможностях *SOAP*-клиентам, которые хотят использовать сервер.

Для данного *SOAP*-сервера формальную *WSDL* спецификацию можно получить, если добавить имя интерфейса после /*WSDL* в адресной строке (*URL*). Для интерфейса *IAppServer URL* для получения *WSDL* спецификации будет выглядеть следующим образом: *http*:///*localhost*/*cgi-bin*/*D7DB2CGI.exe*/*wsdl*/*IAppServer*.

Создание клиента на основе SOAP. После этого при наличии *WSDL* можно создать *SOAP*-клиент. Чтобы продемонстрировать кроссплатформеность подхода, используемого в статье, клиент будет развернут в ОС *Linux*. Разработка

клиента проходит в системе *Kylix*. Кроме того, необходимо использовать реальный *IP*-адрес или *DNS* серверной машины, на которой развернут *SOAP*-сервер.

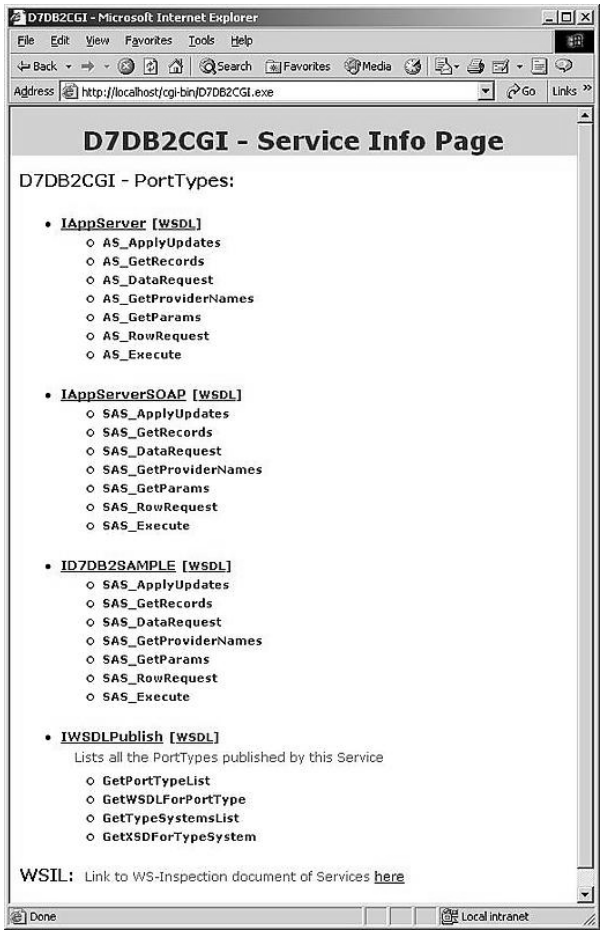

Рис. 7. Сервисная информация для *Web* сервиса *D7DB2CGI*

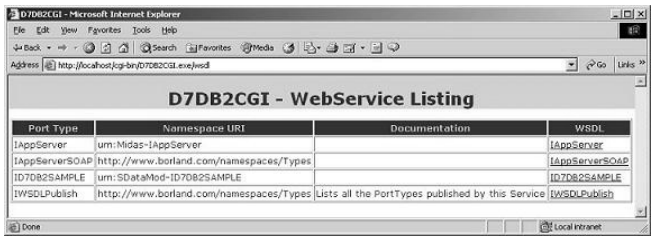

Рис. 8. Листинг *Web*-сервиса для *D7DB2CGI*

• После запуска *Kylix* 3 *Enterprise* необходимо создать новое приложение. Чтобы установить соединение с *SOAP*-сервером нужно разместить на форме нового приложения компонент *TSOAPConnection*, который находится на вкладки *Web Services*. В свойстве *URL* этого компонента необходимо установить значение *http*://192.168.34.101/*cgi-bin*/*D7DB2CGI.exe*/*soap*/*IAppServer*, где 192.168.34.101 – *IP*-адрес машины, на которой развернут *SOAP*-сервер.

• Кроме того, на форме должны быть размещены четыре компонента *TClientDataSet*. Их имена: *cdsEMP*, *cdsEMP_ACT*, *cdsEMP_PHOTO* и *cdsEMP_RESUME* соответственно. Кроме них, нужны еще четыре компонента *TDataSource* с именами *dsEMP*, *dsEMP_ACT*, *dsEMP_PHOTO* и *dsEMP_RESUME* соответственно. После размещения последних необходимо установить связь между ними и компонентами *TClientDataSet*.

• После запуска редактора полей на компоненте *cdsEMP* двойным щелчком мыши, правым щелчком внутри редактора полей нужно вызвать контекстное меню и выбрать «Добавить все поля». В результате станут доступны не только все поля из таблицы *Employee*, но и три специальных поля – *SQLtblEMP_RESUME*, *SQLtblEMP_PHOTO* и *SQLtblEMP_ACT*.

• После добавления устойчивых полей, следует установить свойства *DataSetFields*, чтобы гарантировать, что три детализированных компонента *TClientDataSet* соединились через главный набор данных *cdsEMP*:

- в компоненте *DataSetField* значение свойства *cdsEMP_ACT* необходимо изменить на *cdsEMPSQLtblEMP_ACT*;
- в компоненте *DataSetField* значение свойства *cdsEMP_PHOTO* необходимо изменить на *cdsEMPSQLtblEMP_PHOTO*;
- в компоненте *DataSetField* значение свойства *cdsEMP_RESUME* необходимо изменить на *cdsEMPSQLtblEMP_RESUME*.

Затем на форме необходимо разместить кнопку для обновления БД (вставка, правка и удаление записей) с именем *ApplyUpdates* и добавить для нее обработчик событий (рис. 9).

*Procedure TForm*3.*btnApplyUpdatesClick* (*Sender*: *TObject*);

begin cdsEMP.ApplyUpdates(0) *end*;

Отдельно необходимо написать обработчик событий *OnCreate* и *OnDestroy*, чтобы явно открыть компонент *TClientDataSet cdsEMP* (при запуске приложения) и проверить, были сделаны какие-либо изменения при вызове обработчика кнопки *ApplyUpdates* (когда приложение снова будет закрыто).

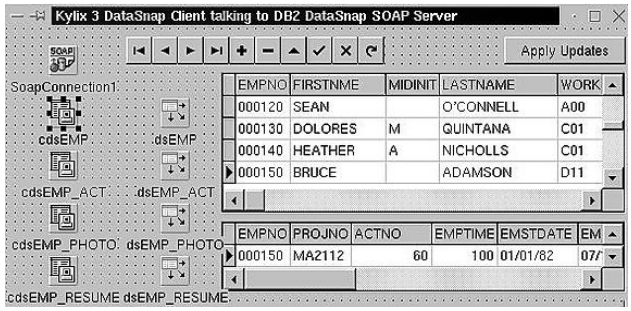

Рис. 9. Форма *SOAP*-клиента на этапе проектирования в *Kylix* 3

*procedure TForm*3.*FormCreate*(*Sender*:*TObject*);
begin cdsEMP.Active := *True end*;
procedure TForm3.FormDestroy(*Sender*: *TObject*); *begin if*
cdsEMP.ChangeCount > 0 *then cdsEMP.Apply Updates*(0); *end*;

После компиляции и запуска приложения, написанного на *Kylix*, *SOAP*-клиент на *Linux* будет иметь следующий вид (рис. 10).

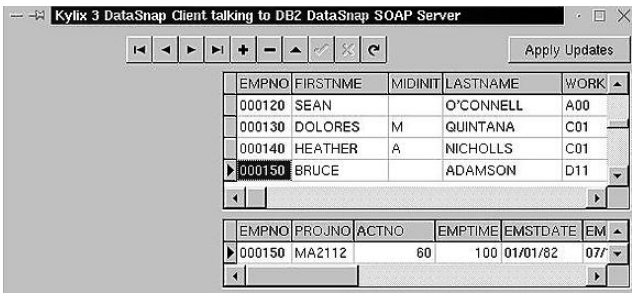

Рис. 10. *SOAP*-клиент в *Kylix* 3

Заключение. Методика, приведенная в статье, позволяет разрабатывать кроссплатформенные приложения, основанные на технологии *Web*-сервисов. Таким образом, КИС с распределенной архитектурой, построенной на базе *Web*-технологий, обеспечивают хорошее масштабирование, понятный и прозрачный процесс администрирования, а также позволяют достаточно просто организовывать удаленный доступ к вычислительным ресурсам, вместе с тем они не требуют специфического клиентского программного обеспечения, т.е. достигается кроссплатформенность всей системы в целом. *Web*-сервисы представляются наиболее подходящим решением для разработки КИС с распределенной архитектурой. Однако существует альтернатива – это семантический *Web* (*Semantic Web*), о необходимости создания которого уже пять лет назад говорил создатель *WWW* Тим Бернерс–Ли. Если задача *Web*-сервисов – облегчить коммуникацию между приложениями, то семантический *Web* призван решить гораздо более сложную проблему – с помощью механизмов метаданных повысить эффективность поиска ценной информации в сети. Сделать это можно, отказавшись от документо-ориентированного подхода в пользу объектно-ориентированного.

1. *Крупин А.Н.* Архитектура информационных систем // Компьютерра. – 2009. – № 12. – С. 39–46.
2. *Марков Е.П.* Архитектура распределенных приложений // Компьютерная неделя. – 2008. – № 15. – С. 102–109.
3. *Батоврин В.К.* Основные направления работ по обеспечению интероперабельности // Третья всерос. конф. «Стандартизация информационных технологий и интероперабельность». – М.: Наука, 2009. – С. 12–15.
4. *Коротенко А.А.*, *Сапегин С.В.* Подходы к интеграции корпоративных информационных систем на основе *sap* XI // Вестн. ВГУ. Сер. Сист. анализ и информ. технол. – 2009. – № 1 – С. 117–121.
5. *Покровский О.Ю.* Анализ архитектур распределенных систем // Перспективные информ. технол. и интел. сист. – 2005. – № 1. – С. 71–78.
6. *Newcomer E.* Understanding Web services: XML, WSDL, SOAP, and UDDI. – Addison–Wesley Professional, 2002. – P. 150–157.
7. *Купцевич Ю.Е.* Альманах программиста, том II: *ASP.NET*, *Web*-сервисы, *WEB*-приложения. – М.: Изд.-торг. дом «Русская Редакция», 2005. – С. 204–207.
8. *Тревис Б. XML* и *SOAP* программирование для серверов *BizTalk*. Новейшие технологии. – Там же, 2007. – С. 212–218.

Поступила 10.11.2011
Тел. для справок: (044) 526-6439, 530-3080 (Киев)
E-mail: *harlam@ukr.net*