ВЕСТНИК ТОМСКОГО ГОСУДАРСТВЕННОГО УНИВЕРСИТЕТА

№ 284 Декабрь 2004

ИНФОРМАТИКА И ПРОГРАММИРОВАНИЕ

УДК 681.3.068

К.Ю. Войтиков, О.А. Змеев, А.Н. Моисеев, А.А. Якушев

АРХИТЕКТУРА НАДСТРАИВАЕМЫХ ПРИЛОЖЕНИЙ КЛИЕНТ/СЕРВЕР С ОБОБЩЕННЫМ ПРОТОКОЛОМ ПЕРЕДАЧИ ДАННЫХ

Предлагаемый в работе подход позволяет проектировать многоуровневые приложения с сильной изоляцией программных компонент и локализацией подсистем, обеспечивающих связь между клиентами и серверами. Данный эффект достигается благодаря введенному в работе понятию «обобщенный протокол передачи данных».

С развитием информационных технологий все большее количество разрабатываемых программных продуктов приобретают статус корпоративных приложений. Такие программные системы являются большими и сложными [1]. Это означает, что разработчики должны постоянно следить за всем имеющимся кодом, контролировать изменения и т.п. – делать все, что входит в понятие «управление проектом». В таком контексте критически важным являются вопросы устойчивости [2] архитектуры разрабатываемой программной системы – ее способности в целом противостоять технологическим изменениям и дополнению функциональности.

В настоящей работе предлагается каркас устойчивой архитектуры многоуровневых приложений клиент/сервер. Этот эффект достигается благодаря сильной изоляции технологических элементов связи между программными модулями системы и механизмов, поддерживающих логику пакетов передаваемых данных. Такая изоляция достигается благодаря введению понятия «обобщенный протокол передачи данных» и решения вопросов, связанных с его реализацией, что позволяет локализовать не только вопросы использования различных средств связи и логики пакетов данных, но и возможности переключения с одного на другой даже во время работы системы.

ПОНЯТИЕ НАДСТРАИВАЕМОЙ СИСТЕМЫ

Рассмотрим задачу создания программной системы с надстраиваемой функциональностью. Под надстраиваемостью будем понимать возможность подключения к готовой функционирующей системе законченных программных модулей, имеющих общее объектное пространство с системой. Понятие надстраиваемости напрямую связано с понятием устойчивости архитектуры [2] программной системы. В нашем случае речь идет о системах, новая функциональность которых приобретается за счет создания новых программных модулей без какой-либо модификации или доработки уже функционирующих.

Рассмотрим общий каркас трехуровневых клиент/серверных приложений (рис. 1).

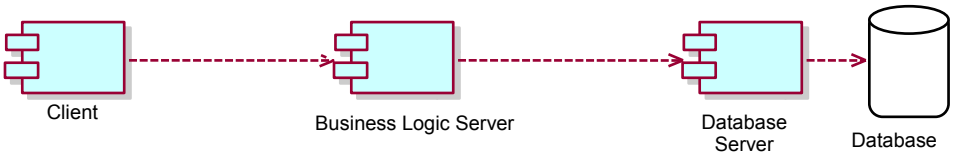

Рис. 1. Компоненты трехуровневого клиент/серверного приложения

Такие системы базируются на трех программных модулях (исполняемых файлах): Client – клиентское приложение, Business Logic Server – сервер бизнес-логики и Database Server – сервер базы данных. Взаимодействие между приложениями происходит на уровне единого пространства имен и команд посредством передачи пакетов данных по определенному протоколу передачи. Общая схема взаимодействия программных модулей выглядит следующим образом: клиент посылает запрос серверу бизнес-логики, который (возможно, после некоторых действий) организует запрос серверу базы данных, а затем (опять же – после некоторых действий), если необходимо, возвращает ответный пакет клиенту.

Поскольку каждый компонент такой системы является достаточно независимым приложением и с некоторой степенью точности может проектироваться самостоятельно, то в дальнейшем будем использовать термин «архитектура» как для понятия общей архитектуры всей системы, так и для ее частей, соответствующих отдельным компонентам или подсистемам. В первом случае будем использовать термин «общая архитектура (системы)», а во втором – «локальная архитектура подсистемы (компонента)».

Следуя поставленной цели создания общей архитектуры «надстраиваемой» программной системы, можно утверждать, что, имея определенную локальную архитектуру ее компонентов, добавление нового модуля не повлияет на общую функциональность системы. Таким модулем может быть как клиент сервера бизнес-логики, так и новый сервер, который будет занимать место между сервером бизнес-логики и клиентом либо между сервером бизнес-логики и сервером базы данных. Более того, этот сервер может быть частично внешним, т.е. иметь собственных клиентов или собственный сервер базы данных (либо не иметь его вообще). В таком случае обычно говорят о четырех- и более звенной архитектуре клиент/сервер. При этом количество добавляемых «промежуточных» серверов или клиентов не должно быть ограничено, а расположены они могут быть на различных узлах локальной или внешней сети.

ОБОБЩЕННЫЙ ПРОТОКОЛ ПЕРЕДАЧИ ДАННЫХ

Основное предназначение клиент/серверных систем – это обмен информацией между клиентским и серверным приложениями. Находясь на разных узлах, они отсылают друг другу информационные пакеты, совершенно не заботясь о том, как они дойдут до адресата. Это является задачей служб, обеспечивающих выход в локальную или

глобальную сети. Любой пакет, отправленный этой службой, преобразуется к виду, соответствующему протоколу передачи данных, по которому взаимодействуют приложения. В настоящее время существует достаточно много способов передачи данных. В данной работе акцентируем внимание на самих пакетах данных, но для начала выделим основные типы протоколов коммуникаций между приложениями.

1. Объектный (к данному типу относятся механизмы COM/DCOM, CORBA). Обмен данными при таком протоколе передачи для разработчика выглядит как обращение к определенным методам интерфейсных объектов сервера. Передача данных на сервер производится через входные параметры этих методов, а возврат результата – через выходные. В случае использования объектных механизмов разработчик должен в коде программы (статически) фиксировать имена (идентификаторы) сервисных объектов и их методов, а также, чаще всего, в определенном порядке перечислять параметры вызова.

2. Двоичный (TCP/IP). Данная группа протоколов поддерживает передачу произвольных данных, обязуя разработчика программно обрабатывать формирование и расшифровку пакетов данных. Чаще всего при этом пакеты соответствуют некоторым внутренним для системы структурам данных или их массивам, в которых важны не только сами данные, но и их порядок.

3. Текстовый (например формат XML). Протоколы этого вида обычно функционируют на основе двоичных и отличаются только тем, что пакет имеет четкую символьную структуру. Обычно такие пакеты могут быть прочитаны «как есть», т.е. вполне понятны для чтения человеком. Порции данных в таких пакетах обычно описываются макроопределениями, вложенными в сам пакет, а порядок их следования не играет роли. В данном случае для формирования и расшифровки пакета программным системам достаточно знать только идентификаторы макроопределений.

Отсюда видим, что основой передачи информации между приложениями кроме транспортных механизмов являются пакеты данных. На первый взгляд, исключение составляют объектные механизмы коммуникаций. Однако для того, чтобы построить более гибкую систему, позволяющую сделать независимым ядро приложения от вида связи и даже иметь возможность изменять его во время выполнения приложения, все функции коммуникаций между модулями следует выделить в отдельную, достаточно изолированную подсистему. Таким образом, актуальной для любых видов протоколов является локальная архитектура подсистемы построения и распознавания пакетов данных.

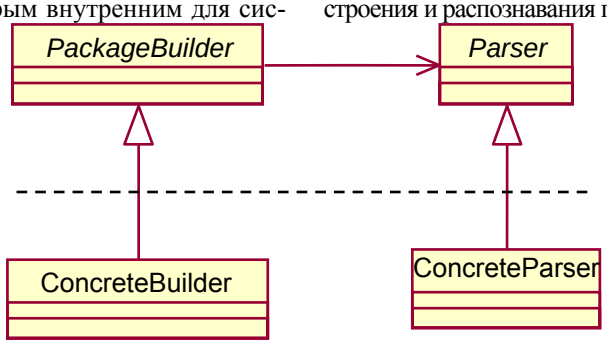

Рис. 2. Подсистема построения пакетов данных

Мы предлагаем архитектуру данной подсистемы, реализованную по шаблону «Строитель» [3]. На диаграмме (рис. 2) изображены следующие классы: PackageBuilder (абстрактный) – построитель пакетов, не включающий реализации каких-либо методов, кроме связи с ассоциированным с ним объектом класса Parser. Последний реализуется по шаблону «Абстрактная фабрика» [3] и включает описание общих методов, обеспечивающих создание пакетов, добавление в них новых элементов и считывание информации. Класс Parser так же является абстрактным и, следовательно, не содержит реализации этих данных методов. Горизонтальной чертой отделены уровень интерфейсов (сверху) и уровень их конкретной реализации (снизу).

Основное назначение классов-потомков PackageBuilder – формирование логики пакета («что должно входить в пакет»), а классов-потомков Parser – поддержка технологического уровня («как это записать в пакет»). В частности, класс ConcreteParser (рис. 2) представляет интерфейс Parser с реализацией методов обработки данных конкретного формата: двоичных, текстовых или даже конкретных текстовых (например, XML с собственным описанием типа документа). ConcreteBuilder – класс, реализующий методы, позволяющие с помощью какого-либо объекта Parser, строить и распознавать пакеты данных на уровне макроопределений, не заботясь о технологической базе пакета.

Поскольку данный подход применим при использовании любых механизмов передачи и форматов самих данных, назовем его обобщенным протоколом передачи данных. Вопросы же самих механизмов связи будут обсуждаться ниже.

НАДСТРАИВАЕМАЯ АРХИТЕКТУРА КЛИЕНТ/СЕРВЕР

Рассмотрим архитектуру клиент/серверных приложений с использованием обобщенного протокола передачи данных (рис. 3). Здесь пунктирными линиями разделены различные подсистемы. Причем движение по диаграмме сверху вниз соответствует переходам от уровня описания интерфейсов к уровням их реализации. Часть, касающаяся объектов Parser, для упрощения опущена, так как, как было описано выше, связь с ними поддерживают непосредственно объекты PackageBuilder.

Любые приложения, использующие обобщенный протокол передачи данных (как клиент, так и сервер), должны иметь не только механизмы формирования пакетов данных, но и средства связи между приложениями, а также подсистемы управления внутренними объектами. В данной архитектуре выполнение указанных задач возлагаем на объекты PackageBuilder, ClientConnection/ServerConnection

и Controller соответственно. Объекты класса Controller реализуются по шаблону «Контролер» [4].

Основные функции объекта ClientConnection – установка соединения с сервером, передача исходящих и прием входящих (ответных) пакетов данных. Для объекта ServerConnection – это прием входящих пакетов от клиента и отправка исходящих (ответных) пакетов. Эти функции заранее определены и не требуют изменений при надстраивании системы.

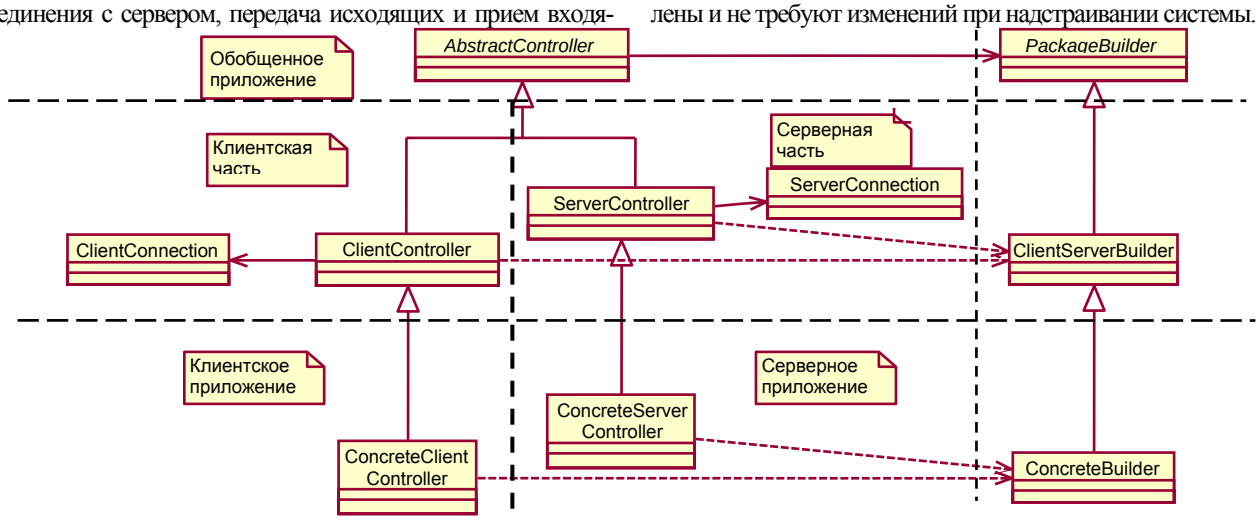

Рис. 3. Архитектура надстраиваемых клиент/серверных приложений с обобщенным протоколом передачи данных

Таким образом, функциональность по управлению внутренними объектами программных модулей и формированию пакетов предлагается вынести на уровень «Приложение общего вида», так как она является общей для построения программных модулей любого вида. Функциональность связи и специфические методы управления объектами в клиентских и серверных приложениях разделяются и выносятся на уровни «Клиентская часть» и «Серверная часть» соответственно.

Таким образом, три описанных уровня (компонента) предоставляют нам каркас клиент/серверной программной системы. На его основе, дополняя подсистемы управления объектами (ClientController, ServerController) и подсистему построения пакетов данных (PackageBuilder), мы можем построить программную систему с произвольной клиент/серверной структурой (см. пример далее). В частности, на диаграмме показаны программные модули ConcreteClient и ConcreteServer, реализующие некоторые конкретные приложения клиента и сервера, а также объект ConcreteBuilder, необходимый для поддержки единого пространства макроопределений пакетов.

ПРИМЕР РЕАЛИЗАЦИИ ТРЕХУРОВНЕВОГО ПРИЛОЖЕНИЯ

Рассмотрим классическую модель трехуровневого клиент/серверного приложения типа «клиент – сервер бизнес-логики – сервер базы данных». На диаграмме (рис. 4) изображены компоненты, составляющие основу такого приложения, а также объекты, входящие в их состав. Рассмотрим эту диаграмму подробнее.

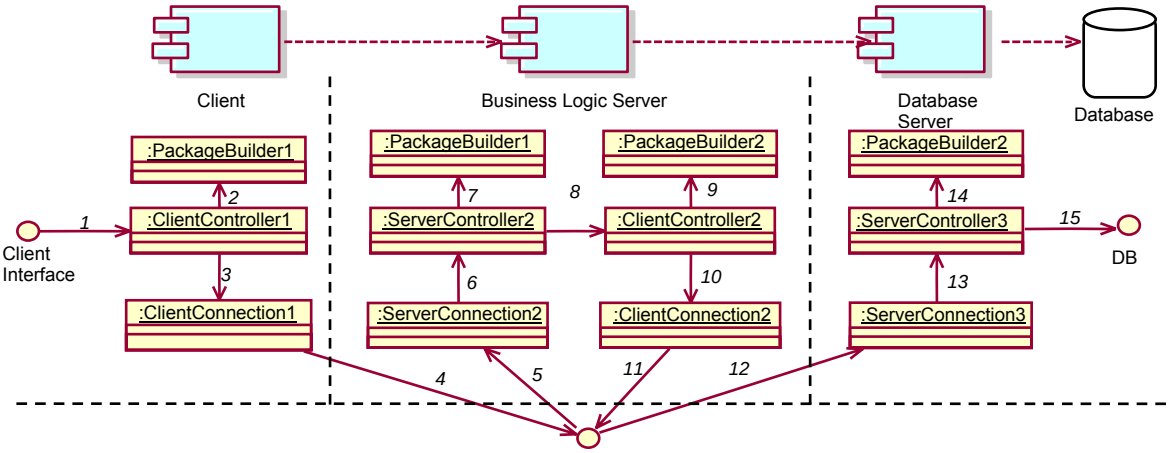

Рис. 4. Пример использования каркаса архитектуры надстраиваемых клиент/серверных приложений с обобщенным протоколом передачи данных

Для того чтобы акцентировать внимание на общих и различающихся частях отдельных приложений системы, используем дополнительное цифровое обозначение в именах классов. Причем число 1 будет соответствовать компоненту «Клиент», 2 – компоненту «Сервер бизнес-логики», 3 – компоненту «Сервер базы данных». Например, ClentController2 – класс, отвечающий функциональности класса ConcreteClientController (рис. 3) для компонента «Сервер бизнес-логики». Имена некоторых классов в разных компонентах совпадают (например, PackageBuilder1 в компонентах «Клиент» и «Сервер бизнес-логики»). Это означает, что соответствующие объекты в точности реализуют одинаковую функциональность. Почему это так, объясняется ниже.

При необходимости выполнить действия, связанные с обращением к серверу бизнес-логики, клиентское при-

ложение через интерфейс (Client Interface) передает сообщение (обозначено числом «*1*») объекту :ClientController1. Последний, с помощью объекта :PackageBuilder1, строит исходящий пакет данных (*2*) и передает его объекту :ClientCon-nection1 (*3*). Далее объект :ClientConnection1 посредством выбранной внешней службы связи передает этот пакет серверу бизнес-логики (*4*, *5*). Объект :ServerConnection2 передает входящий пакет объекту :ServerController2 (*6*), который с помощью объекта :PackageBuilder1 расшифровывает принятый пакет данных (*7*). Обратите внимание на то, что клиентское приложение и сервер бизнес-логики используют построители пакетов одного и того же класса (PackageBuilder1). Это необходимо для поддержки идентичности макроопределений пакетов.

Далее, возможно, после некоторых действий, объект :ServerController2 передает управление объекту :ClientController2 (*8*). То есть в данной ситуации сервер бизнес-логики выступает в роли клиента сервера базы данных :ClientController2 с помощью объекта :PackageBuilder2 формирует исходящий пакет с запросом к серверу базы данных (*9*) и отправляет его посредством объекта :ClienConnection2 (*10*, *11*, *12*). Здесь специально указан другой класс объекта ClientConnection, так как связь между сервером бизнес-логики и сервером базы данных может производиться с использованием средств коммуникаций, отличных от средств, использующихся для связи клиента с сервером бизнес-логики.

Затем объект :ServerConnection3 передает входящий пакет данных объекту :ServerController3 (*13*), который, используя объект :PackageBuilder2, расшифровывает принятый пакет (14) и реализует запрос к базе данных (*15*). Здесь также использован принцип одного класса построителя пакетов для двух взаимодействующих приложений (PackageBuilder2). Теперь проанализируем надстраиваемость архитектуры данного приложения. Для этого рассмотрим различные ситуации, когда к системе добавляется новый программный модуль.

1. Добавляется клиент сервера бизнес-логики (рис. 5). Соответствующий компонент должен иметь базовую локальную архитектуру, аналогичную первому клиенту. Поэтому при обращении к серверу бизнес-логики происходят те же действия, что и рассмотренные выше. Нет необходимости производить какие-либо изменения в архитектуре системы.

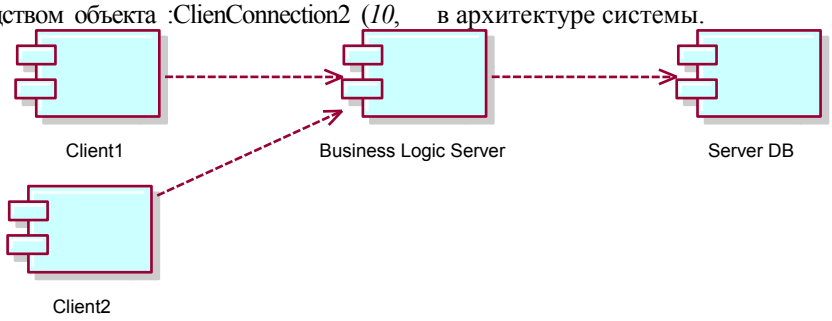

Рис. 5. Добавление к системе нового клиента сервера бизнес-логики

2. Добавляется клиент сервера бизнес-логики, выступающий в роли сервера для других приложений (рис. 6). Такой программный модуль должен включать в себя структуры, соответствующие как локальной архитектуре серверной части (для клиентов), так и локальной архитектуре клиентской части (для сервера бизнес-логики). Такое построение аналогично структуре сервера бизнес-логики. В данном случае система соответствует типичной четырехуровневой модели клиент/сервер.

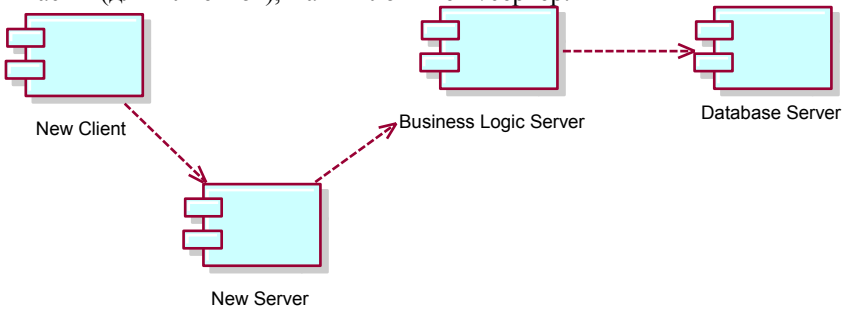

Рис. 6. Добавление к системе нового сервера, являющегося клиентом сервера бизнес-логики

3. Добавляется новый сервер бизнес-логики параллельно уже существующему (рис. 7). Оба сервера бизнес-логики обращаются к одному серверу базы данных. Клиенты серверов бизнес-логики могут обращаться как к одному из них, так и к двум одновременно. Структура нового сервера бизнес-логики аналогична существующему, а клиентское приложение в случае различий в пространствах макроопределений серверов должно включать в себя еще один объект PackageBuilder и, возможно, ClientConnection – в случае новых механизмов связи.

4. Добавляется новый сервер бизнес-логики, который является клиентом уже существующего сервера бизнес-логики (рис. 8). Архитектура системы аналогична случаям 2 и 3 только с учетом того, что новый сервер бизнес-логики обращается к существующему и не связан с сервером базы данных.

5. Добавляется новый сервер бизнес-логики, который может быть клиентом уже существующего сервера бизнес-логики и сервера базы данных (рис. 9). Архитектура аналогична предыдущей ситуации, к тому же новый сервер бизнес-логики включает в себя необходимые подсистемы для связи с сервером базы данных.

6. Добавляется новый сервер бизнес-логики и сервер базы данных, работающие параллельно с сущест-

вующими модулями (рис. 10). Очевидно, что и в этом случае архитектура системы не претерпевает глобальных изменений и легко реализуется на предлагаемом в данной работе каркасе.

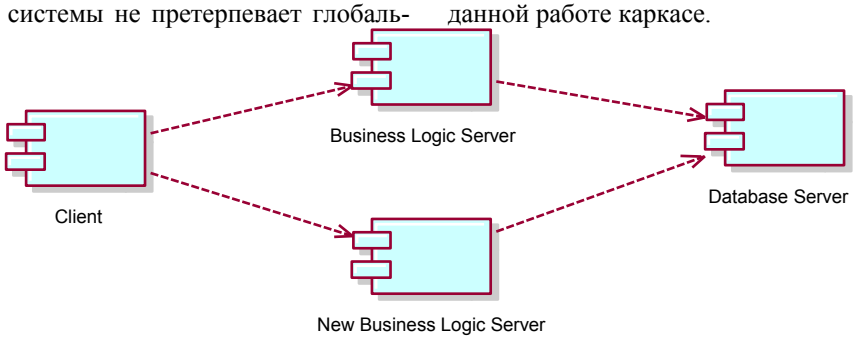

Рис. 7. Добавление к системе нового сервера бизнес-логики, параллельно существующему

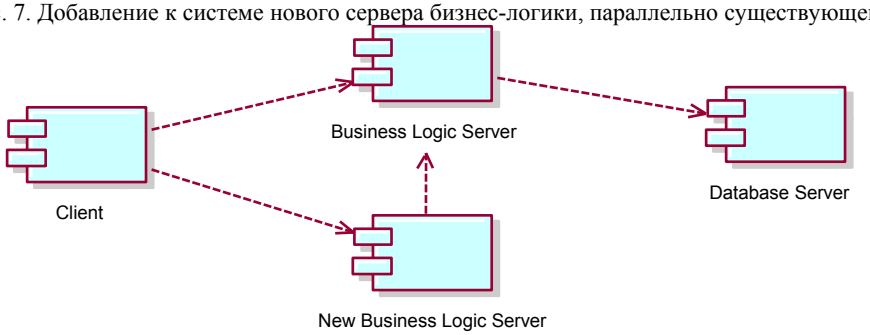

Рис. 8. Добавление нового сервера бизнес-логики,
который является клиентом уже существующего сервера бизнес-логики и сервером уже существующего клиента

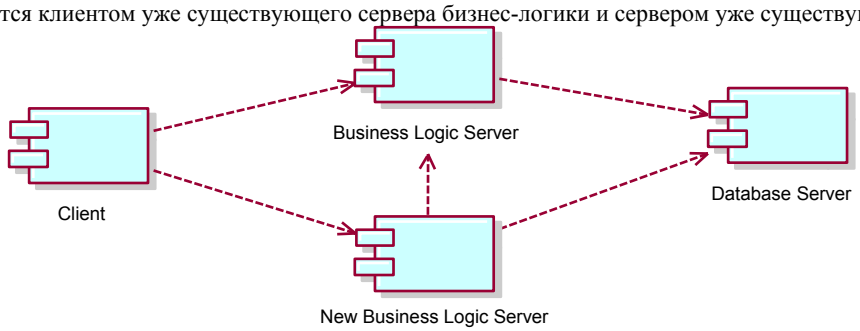

Рис. 9. Добавление нового сервера бизнес-логики,
который может быть клиентом уже существующего сервера бизнес-логики и сервера базы данных

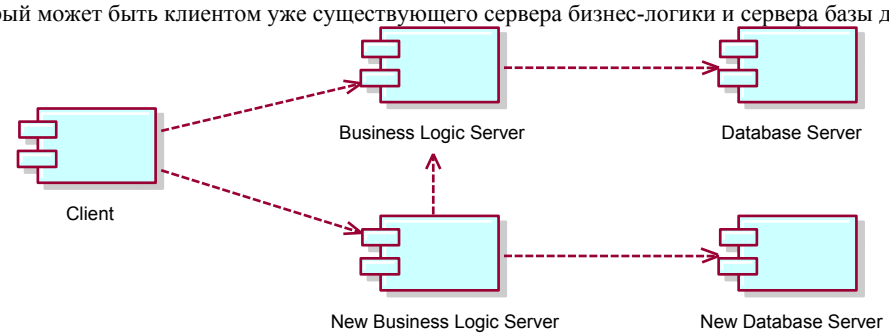

Рис. 10. Добавление новых серверов бизнес-логики и базы данных, параллельно существующим

ВЫВОДЫ

Предложенный в работе каркас надстраиваемого объектного многоуровневого приложения с обобщенным протоколом передачи данных позволяет проектировать программные системы с устойчивой архитектурой. При этом на архитектуру построенных приложений вновь вводимая функциональность, оформленная в виде новых компонент (клиенты, серверы), оказывает локальное влияние. К тому же подход обобщения протокола передачи данных позволяет локализовать как изменения в технологиях связи, так и в пространствах макроопределений пакетов данных.

ЛИТЕРАТУРА

1. *Буч Г.* Объектно-ориентированный анализ и проектирование с примерами приложений на C++. СПб.: Невский проспект, 2001. 560 с.
2. *Якобсон А., Буч Г., Рамбо Дж.* Унифицированный процесс разработки программного обеспечения. СПб.: Питер, 2002. 496 с.
3 *Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж.* Приемы объектно-ориентированного проектирования. Паттерны проектирования. СПб.: Питер, 2001, 368 с.
4 *Ларман К.* Применение UML и шаблонов проектирования. 2-е изд. М.: Изд. дом «Вильямс», 2002. 624 с.

Статья представлена кафедрой теоретических основ информатики факультета информатики Томского государственного университета, поступила в научную редакцию «Информатика» 30 апреля 2004 г.