Детальное проектирование и конструирование

Детальное проектирование включает в себя проработку решений, принятых при проектировании архитектуры. Разрабатываются внутренние подсистемы, сервисы, форматы, протоколы и т.п. и параллельно относящаяся к ним структура кода.

Далее будут рассмотрены два реальных и два гипотетических (потенциально реализуемых в данной архитектуре при преемственности программного кода) варианта прикладного протокола клиент-сервер.

Структура программного кода

Для удобства повторного использования кода значительная часть классов выделена в **ядро (core)** - уровни контроллера (приложение - “преподаватель”), бизнес-объектов, табличных объектов, коммуникации, коннекторы к БД, интерфейсы

Компонента	Пакет	Классов	Строк	Назначение
Core (ядро)	brs	2	325	Параметры настройки приложений
	brs.connect	4	828	Коннекторы к БД
	brs.model	29	3412	Бизнес-объекты
	brs.database	34	1207	Табличные объекты БД
	brs.ftpclient	4	880	ftp-клиент хранилища - не используется
	brs.controller	3	646	Контроллер бизнес-логики "преподаватель"
	bts.xml	8	244	Команды-ответы протокола WebAPI
	brs.interfaces	5	143	Интерфейсы компонент
	brs.cui	5	131	Объекты протокола ЦИУ
Android	brs.view	26	3755	Уровень представления (экраны)
	brs.sqlite	2	289	Коннектор БД SQLite (Android) - не используется
Desktop (все приложения)	brs.javaview	40	7748	Уровени представления и бизнес-логики (экраны) - все приложения, "преподаватель" - только представление
	view	1	232	Таблица с прокруткой
WebAPI	brs.web	1	684	Сервлет WebAPI для тонкого клиента

Рис. 5.4. Структура программного кода

Классы табличных объектов БД - толстый клиент

Формально толстый клиент создается на основе библиотеки-коннектора к серверу БД MySQL (JDBC), которая допускает в том числе и дистанционный доступ через сеть. Но SQL-запросы не используются на уровне бизнес-логики. Единственным «пользователем» коннектора являются объекты уровня доступа к данным (DAO). Все они являются наследниками класса DBItem, в котором реализованы следующие возможности:

- преобразование (с использованием рефлексии Java) списка полей и их значений в табличном объекте в SQL-запрос на его выбор, добавление или обновление в таблице БД, а также создание самой таблицы;
- единая система именования идентификаторов и ссылочных полей;
- получение связанных записей по значениям ссылочных полей.

Классы табличных объектов соответствуют основнным классам-сущностям модели предметной области, а ссылочные поля – отношениям между классами.

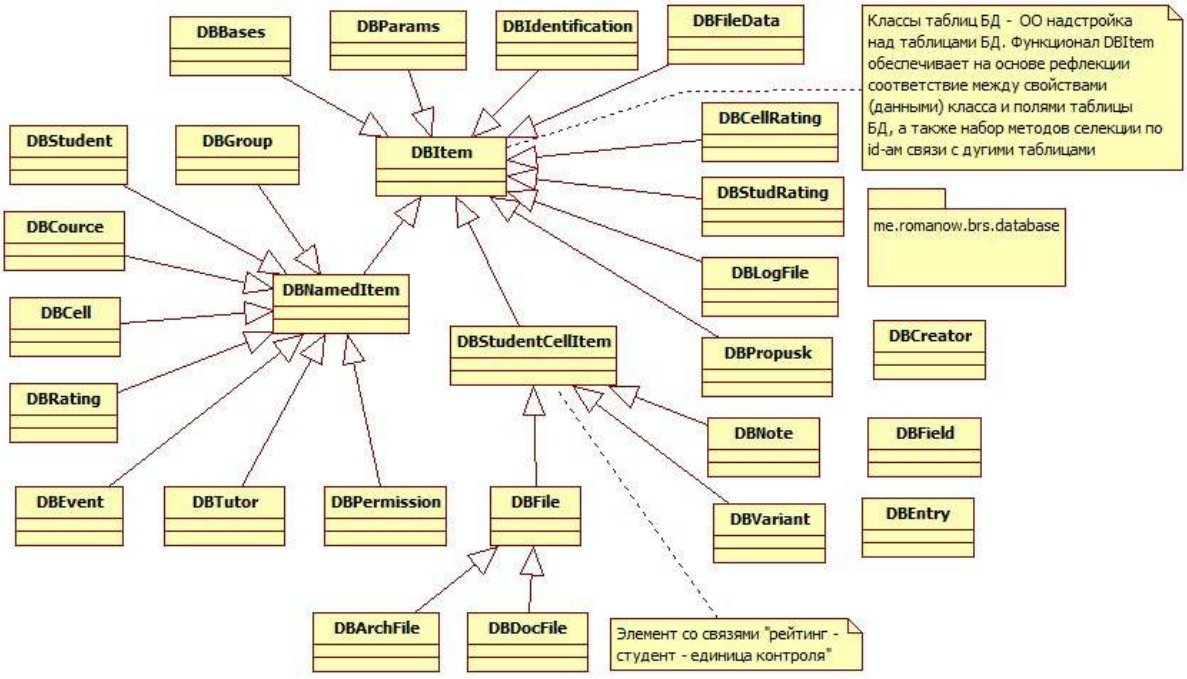

Рис. 5.5. Диаграмма классов таблиц БД

Структура БД, изображенная ниже, также генерируется автоматически по спискам полей данных классов, производных от DBItem.

Group	Rating	Event *	Entry (Profile)**	Variant (x)
id	id	id	id	id
name	name	name	name	idStudent
	idGroup	idRating	user	idRating
	idCourse	evtDate	pass	idCell
Student	second	**removed**	ip	**cDate**
id	idParams		port	variant
name		**Propusk ***(x)**	selected	
idGroup	**Params**	id		**DocFile/ArchFile (x)**
ciuLogin	id	idStudent	**Tutor**	id
pass	maxWeek	idRating	id	idStudent
	weekProc	idEvent	name	idRating
Course	propuskBall	**removed**	pass	idCell
id	plusProc	**cDate**	ciuName	**cDate**
name	semestr			fileId
	tutorId		**Permission**	fileName
Cell			id	
id	**StudRating (*)**	**CellRating (*)**	name	**Note ***(x)**
name	id	id	idTutor	id
idCourse	idStudent	idRating	idRating	idStudent
cType	idRating	idCell		idRating
ball	brigade	week	**FileData**	idCell
ordNum	second	week2	id	ball
	cDate	**cDate**	data	week
			idFileData	params
				removed
				cDate

* - заполняются при добавлении рейтинга по его элементам
** - файл brs_config.dat
*** - множественные экземпляры записей для архива операций
x - избыточное поле idRating для восстановления контекста

Рис. 5.6. Структура БД

Классы бизнес-объектов – тонкий клиент на основе WebApi

Классы бизнес-объектов составляются из связанных табличных объектов таким образом, что следующий уровень бизнес-процессов получает уже готовые «сборки» бизнес-сущностей как структур данных из связанных объектов. Бизнес-объекты имеют наборы базовых методов и несколько уровней “развернутости” (краткий, базовый и полный). Основной бизнес-объект - рейтинг, инкапсулирующий в себя все остальные.

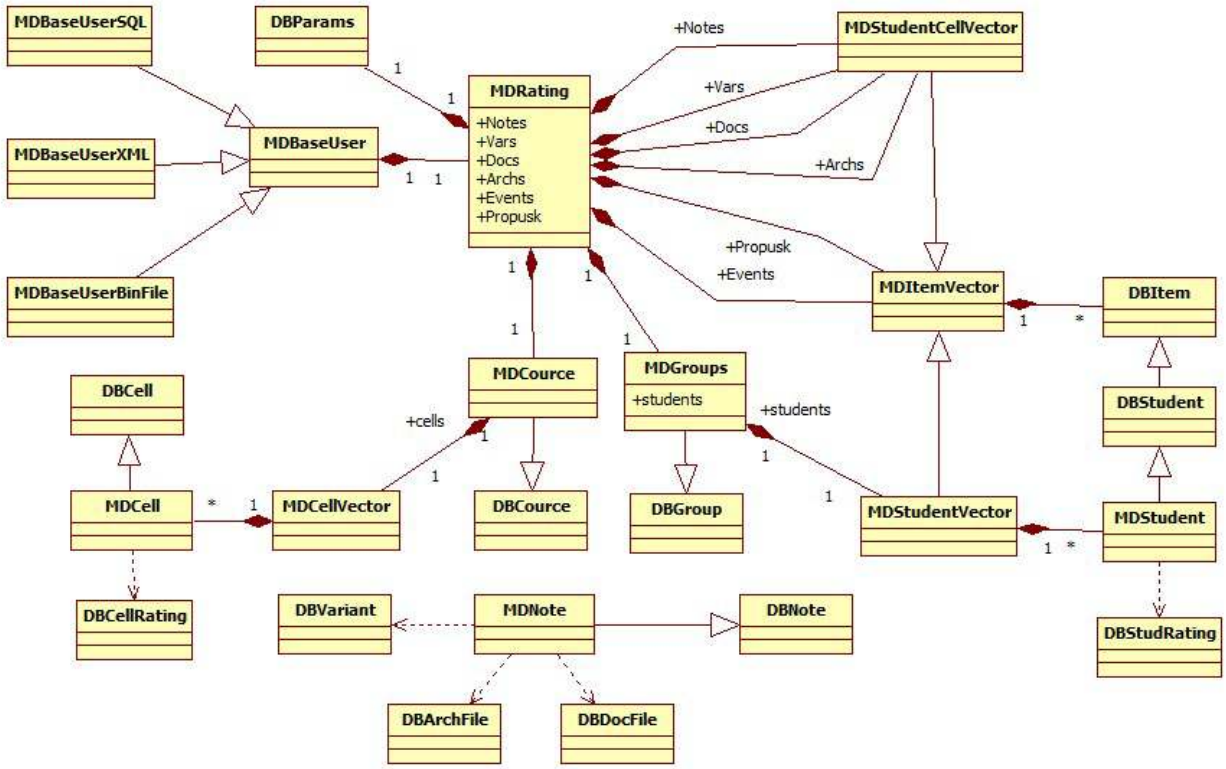

Рис. 5.7. Диаграмма классов бизнес-модели (бизнес-объекты)

Прикладной протокол этого слоя – протокол «умеренно» тонкого клиента с использованием WebAPI – подробно описан в 4.4. Здесь кратко напомним архитектурное решение, позволяющее максимально объединить код при реализации двух вариантов толстого и тонкого клиента.

Классы коммуникаций

Классы коммуникаций обеспечивают реализацию толстого и тонкого (WebAPI) клиентов. Базовый класс бизнес-логики **MDBaseUser** имеет 3 расширения:

- **MDBaseUserSQL** загружает бизнес-объекты непосредственно через соединение с БД, используется в толстом клиенте
- **MDBaseUserBinFile** сохраняет бизнес-объект “рейтинг” в двоичном файле, используется для создания локальных копий рейтингов

- **MDBaseUserXML** реализует тонкого клиента с использованием WebAPI. Используется ряд классов, сериализуемых в XML, для поддержки команд и ответов протокола (XMLCmd, XMLAnswer, XMLArray). В протоколе также сериализуются бизнес-объекты (MDNote). MDBaseUserXML является клиентской частью протокола, а **MDXMLCommand** - серверной (Java-сервлет). Последний создает на сервере объект MDBaseUserSQL и выполняет в нем необходимые команды от имени клиента.

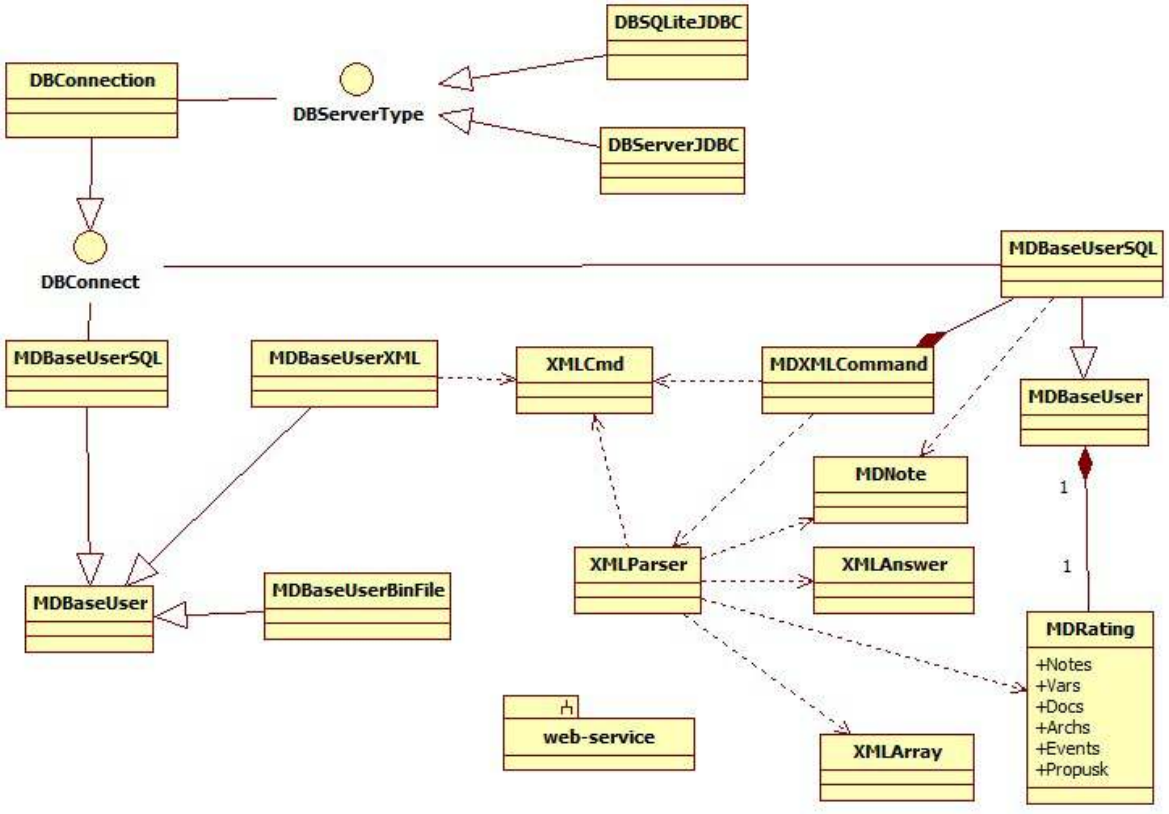

Рис. 5.8. Диаграмма классов коммуникаций

В качестве иллюстрации приведем диаграммы последовательности для толстого и тонкого клиентов. Даже поверхностное визуальное сравнение позволяет увидеть идентичность поведения (на стороне клиента и на стороне сервера в первом и втором случае)

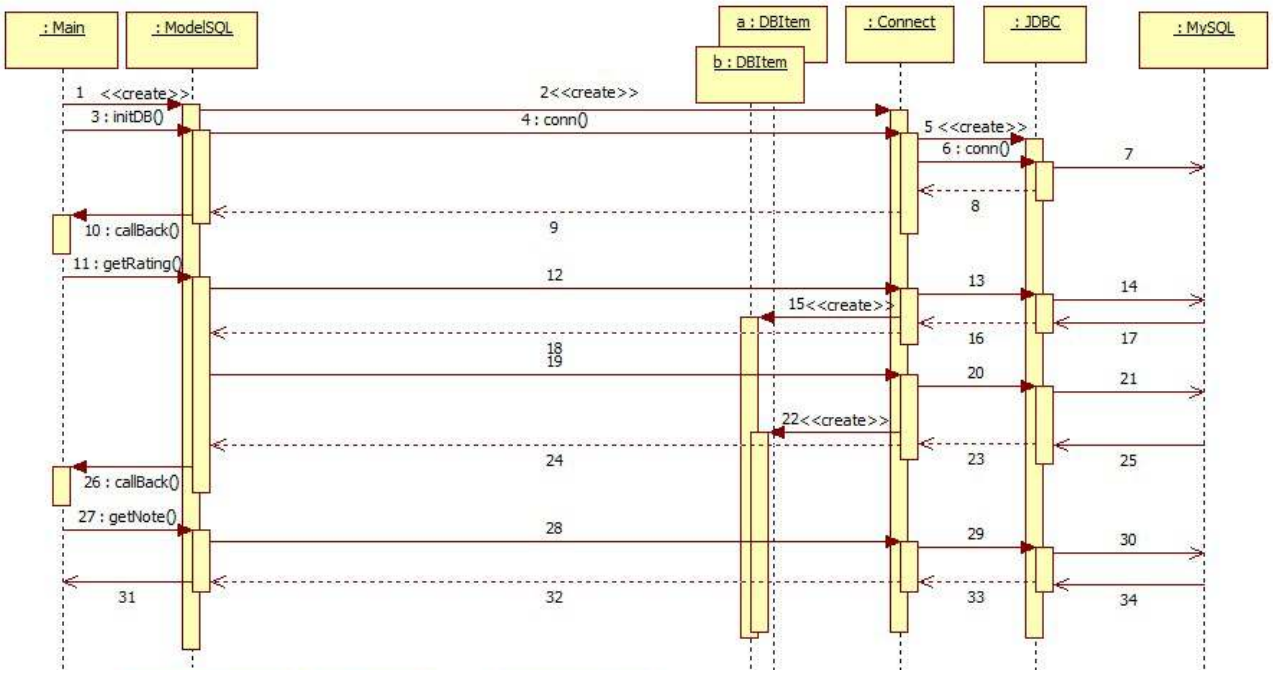

Рис. 5.9. Толстый клиент. Сетевая компонента интегрирована в библиотеку доступа к БД

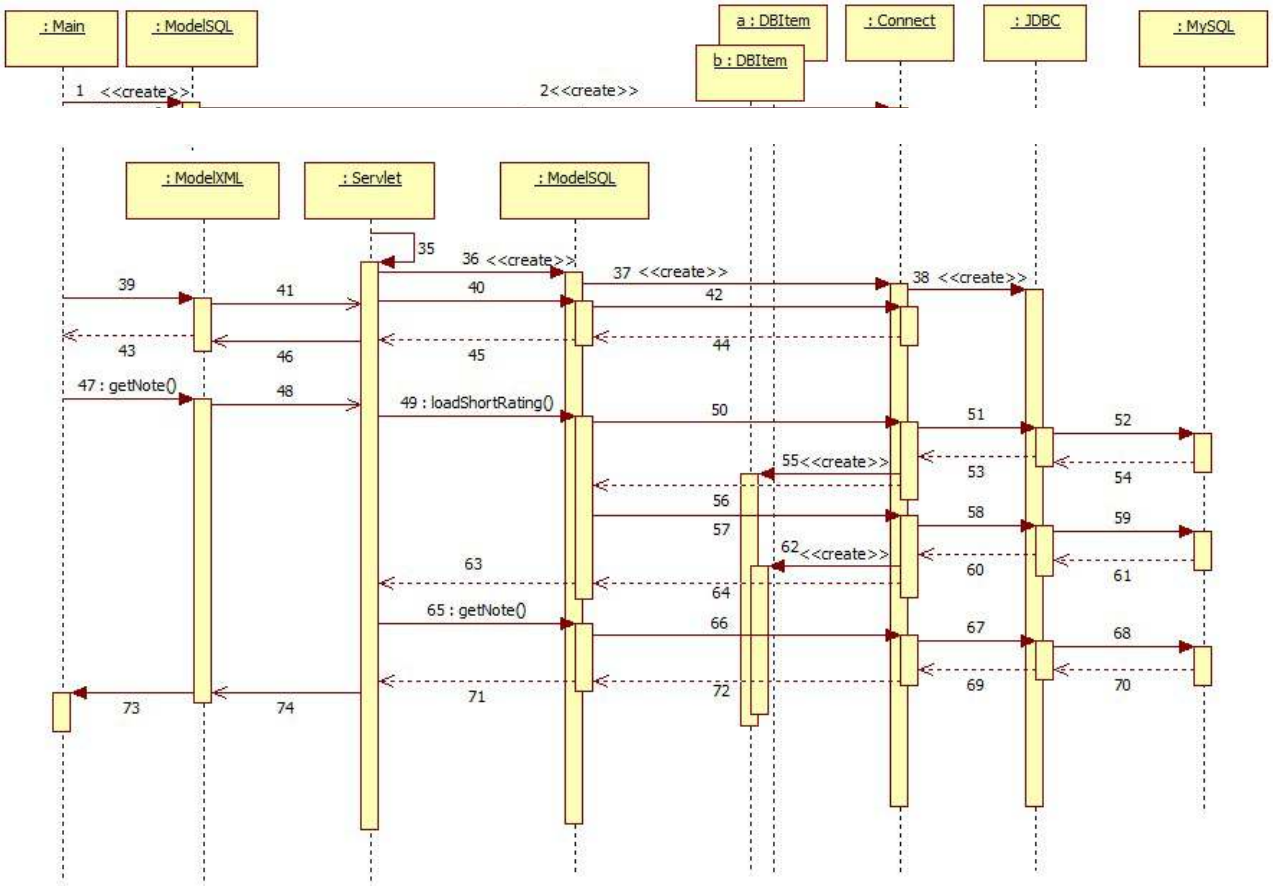

Рис. 5.10.Тонкий клиент. Сетевой протокол реализован в классах ModelXML (MDBaseUserXML) и Servlet (MDXMLCommand)

В приведенной реализации классы протокола для клиента и сервера написаны на Java и используют XML-сериализацию передаваемых сообщений. Поэтому спецификацией протокола можно более-менее однозначно соотнести с программным кодом этих модулей. Тем не менее, формальная спецификация протокола бывает необходима для возможности открытой реализации клиента или сервера сторонними разработчиками. Ниже приводится сокращенный пример спецификации протокола webAPI.

Клиент посылает POST-запрос с фиксированным набором параметров. Параметры со значением 0 не передаются. Кириллица кодируется стандартным URL-кодером (в виде %код символа. Параметры запроса:

- code – код команды
- id,id2,id3,id4 - числовые параметры
- str - строковый параметр

Некоторые команды имеют дополнительные параметры, которые определяются прикрепленным к команде объектом.

Ответ сервера всегда начинается с xml-кадра вида <ans code="-1" message="" id="2"/>, который имеет 3 обязательных параметра. XML-кадры передаются в кодировке Windows-1251.

Значение code=-1 соответствует нормальному ответу сервера, после которого не следует никаких данных. В этом случае параметры message и id могут содержать данные ответа сервера.

Значение code=0 соответствует нормальному ответу сервера, после которого следует xml-кадр, вид которого зависит от команды. В этом случае параметры message и id могут содержать данные ответа сервера.

Значение code>0 соответствует завершению команды с ошибкой, сам code содержит ее код, а message – детализирующее сообщение. Коды ошибок:

- "Ошибка программы (баг)", // 1
- "Ошибка SQL-запроса", // 2
- "Ошибка сети", // 3
- "Ошибка структуры БД", // 4
- "Функция не реализована", // 5
- "Нераспознанная ошибка", // 6
- "Ошибка сервера", // 7
- "Ошибка настройки" // 8

Приведена классификация ошибок во всех компонентах системы. Для BRSWeb возвращается ошибка с кодом 7, в message содержится код ошибки сервлета (1-6,8) и ее детализация.

Список команд-ответов

Принятые по умолчанию имена полей в ответах:

- id – идентификатор объекта в БД
- idg - id студенческой группы
- ids - id студента
- idc - id дисциплины (курса)
- idu - id учебной единицы
- ide - id занятия
- ida - id файла архива
- idd - id файла документа
- idr - id рейтинга
- name - основное имя (название) объекта
- date - дата – целое в кодировке dd+mm*35+yy*35*15
- ball - балл
- var - вариант
- week - номер недели

ping (0) - пустая команда.

Получение данных

getRatingList (1) - список рейтингов (параметры команды и ответ сервера)

```
code=1
<ans code="0" message="" id="0"/>
<list>
  <rating id="1" name="ДГИ АВТ-209" idc="1" idg="4" idp="0" sec="true"/>
  <rating id="2" name="ДГИ АВТ-215" idc="1" idg="5" idp="0" sec="true"/>
</list>
```

testUser(2) – проверка логина/пароля преподавателя (id=idt, id2=0 – локальный пароль, id2=1 – проверка по паролю ЦИУ)

```
code=2&id=1&id2=1&name=vt_romanov&pass=777hhh
<ans code="-1" message="Недопустимый логин/пароль ЦИТ" id="0"/>
code=2&id=1&id2=1&name=vt_romanov&pass555fff
<ans code="-1" message="" id="0"/>
code=2&id=1&name=vt_romanov&pass=gugugu
<ans code="-1" message="" id="0"/>
```

testEdit(3) – проверка на разрешение редактирования рейтинга, не используется.

loadRating(5) – сокращенный рейтинг по его id

loadFullRating(6) – полный рейтинг по id, используется в Android-клиенте для закачки рейтинга и для генерации отчетов по группе (оценки и пропуски)

loadStudentRating(11) и loadCellRating(12) имеют аналогичный формат, но возвращают данные по студенту или учебной единице (id2) (данные по пропускам в **loadCellRating** не селектируются)

loadStudentList=7; – список студентов (рейтинг id) – не используется

loadCellList(8) – список учебных единиц (рейтинг id) – не используется

loadEventList(9) – список событий (рейтинг id)

calcStudentRating(10) – суммарный балл студента id2 в рейтинге id (результат в id)

Редактирование оценок и пропусков

getNote(20) – текущая оценка (рейтинг id, студент id2, учебная единица id3)

getNoteHistory(21) –история редактирования оценки (рейтинг id, студент id2, учебная единица id3)

changeNote(22) –редактировать (рейтинг id, студент id2, учебная единица id3)

changeDocFile(23) – редактировать имя файла документа (рейтинг id, студент id2, учебная единица id3), пустая строка str – документ удален, id4 – идентификатор файла, полученный при записи командой **writeDocFile(60)**

changeArchFile(24) – редактировать имя файла архива – аналогично предыдущему

changeVariant(25) – редактировать вариант задания (рейтинг id, студент id2, учебная единица id3, вариант (строковое значение) в str)

getEvent(26) – данные занятия (пропуски) (рейтинг id, занятие id2)

changeEvent(27) – редактировать пропуски занятия (рейтинг id, занятие id2, студент id3), id4=1 – присутствие, id4=0 – пропуск.

changeBrigade(28) – редактировать номер бригады (подгруппы) (рейтинг id, студент id2), бригада id3, id4=0 – подгруппа 1, id4=1 – подгруппа 2

changeWeek(29) – редактировать срок сдачи) (рейтинг id, учебная единица id2), id3 – номер недели для 1-ой подруппы, id4 – для 2-ой.

insertEvent(30) – добавить занятие (рейтинг id), str-название, id2-дата

deleteEvent(31) – удалить занятие и пропуски ((рейтинг id, занятие id2)

Управление хранилищем файлов

writeDocFile(60) – записать файл документа в хранилище, id – длина файла в байтах, после передачи заголовка следуют данные самого файла в виде потока байтов в количестве, указанном в id. В ответе в id возвращается уникальный идентификатор файла в БД.

writeArchFile(61) – аналогично предыдущему для файла архива

readDocFile(62) – читать файл из хранилища по идентификатору в БД – id, в ответе id – длина файла в байтах, после ответа следует двоичный поток байтов в количестве, указанном в id.

readArchFile(63) – аналогично предыдущему для файла архива

deleteFile(64) – удалить файл из архива по идентификатору в БД – id, id2=0 – архив, id2=1 - документ

Работа с таблицами (администрирование)

insert(80) – добавление записи в таблицу str, вторая XML-запись – объект, не используется

delete(81) – удаление записи из таблицы str по id, не используется

Авторизация

getTutorList(101) – список преподавателей

getGroupsList(102) – список групп

getStudentList(103) – список студентов группы id

testStudent(104) – проверка логина/пароля студента по id, непустой логин – ЦИУ, иначе – локальный пароль.

testBase(105) - сравнение имени открытой сервером БД с передаваемой в запросе. Ответ id=2 – истина, id=1 – ложь.

Классы контроллер - представление – тонкие клиенты (web-клиент и приложение)

Основная цель отделения контроллера от представления в приложении «преподаватель» - необходимость поддержки приложения на двух платформах – desktop и мобильное приложение под Android. Чтобы избежать необходимости «копипаста», требуется, чтобы в представлении не было ни одной строчки кода, соответствующей **поведению** бизнес-модели, а только компоненты отображения.

Контроллер является имитатором поведения абстрактного представления (View). Представление – тонкий клиент, лишенный модели поведения. Контроллер и представление связаны двумя интерфейсами - “событий” в представлении и “команд” контроллера по вводу/выводу содержимого и отображению элементов управления в представлении. Модель - набор бизнес-объектов и бизнес-методов - в основном управляется контроллером, хотя некоторые данные могут быть получены непосредственно представлением.

Контроллер реализует варианты поведения представления, определяет возможные последовательности «нажатия кнопок» и генерирует события для изменения состояния элементов отображения. Контроллер реализует модель поведения на основе диаграммы состояний.

Рис. 5.11. Диаграмма состояний представления (View) в контроллере

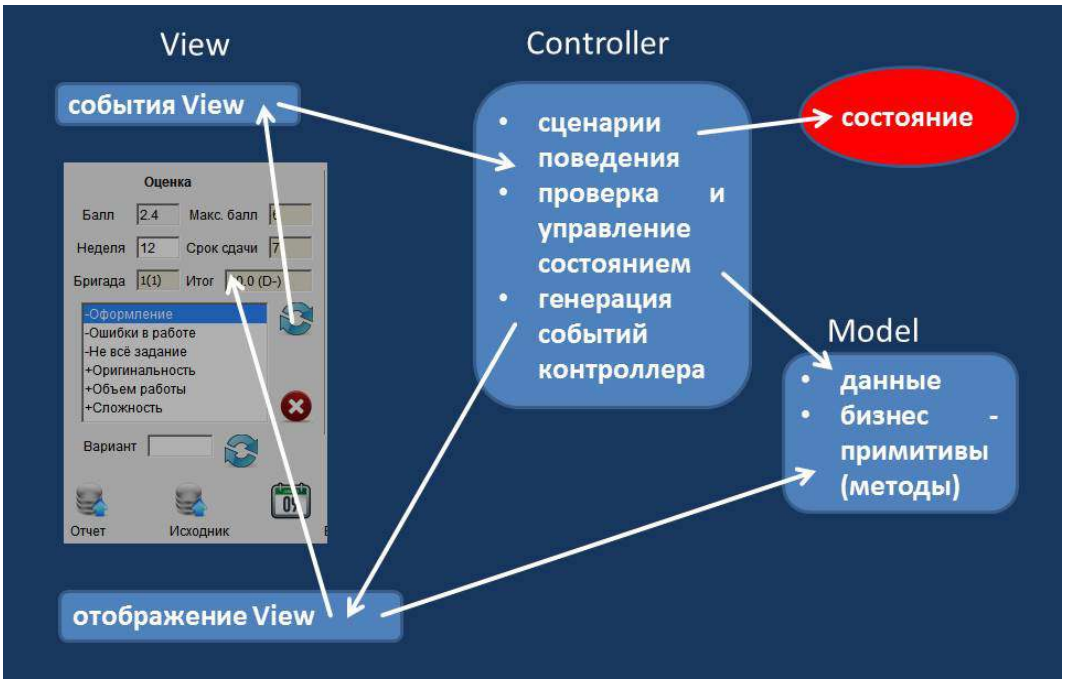

Рис. 5.12. Взаимодействие представления (View) и контроллера

Интерфейс ViewControllerCommands определяет набор команд контроллеру со стороны представления, инициируемых такими событиями как нажатие кнопок или выбор элементов списков.

```
//---------------------------------------------------------------------------------------- ViewControllerCommands.java
public interface ViewControllerCommands {
   public double getSum();
   public MDNote getNoteItem();
   public void setState();
   public void retryState();
   public void close();
   public void updateAllLocalRatings();
   public void loadRating(int id) throws Throwable;
   public void loadFullRating(int id) throws Throwable;
   public void loadFullRatingSoft(int id) throws Throwable;
   public void setState(int newState);
   public void callBacks();
         …
}
```

Интерфейс ViewControllerListener определяет набор команд контроллера для выполнения представлением действий над элементами отображения.

```
//---------------------------------------------------------------------------------------- ViewControllerListener.java
public interface ViewControllerListener {
   public String getInputFileName(String title, String defName) throws Throwable;
   public String getOutputFileName(String title, String defName) throws Throwable;
   public void fatal(Throwable ee);
   public void stateChanged(int state);
   public void setRatingVisible(boolean enb);
   public void setCellVisible(boolean enb);
   public void setStudentVisible(boolean enb);
   public void setNoteVisible(boolean enb);
```

```
    public void setWeekVisible(boolean enb, String s);
    …
}
```

Наличие контроллера, полностью отделенного от представления, позволяет предложить еще два варианта **тонкого клиента.**

Тонкий клиент – приложение. На основе приведенных интерфейсов можно создать двунаправленный протокол обмена сообщениями (см. 4.3 - синхронный, 4.5 - асинхронный). Методы интерфейсов будут соответствовать типам сообщений, а параметры – содержимому. На клиентской стороне будут находиться **представление** и **клиент протокола**. Клиент протокола будет принимать команды ViewControllerCommands, оформлять их в сообщения и передавать на сервер. Принятые сообщения он будет преобразовывать в вызовы ViewControllerListener в представлении. На серверной стороне будет находиться цепочка **агент протокола – контроллер …** и далее все слои вплоть до БД. Агент протокола будет выполнять симметричные действия по отношению к контроллеру. Вся серверная цепочка будет создаваться для каждого соединения своя.

Примечание. Естественно, имеющиеся интерфейсы нуждаются в переработке, т.к. некоторые методы по существу являются синхронными (т.е. возвращают результат или получают заполненные данными объекты с задержкой). Кроме того, система генерации исключений также должна быть «пропущена через протокол», т.к. исключения генерируются на стороне контроллера на сервере.

Тонкий web-клиент. Рассмотренный вариант тонкого клиента, который имеет малую функциональность на клиентской стороне, но в то же время выполнен в виде отдельного приложения, не вполне логичен (подобные решения были жизнеспособны, например, в мобильных телефонах с поддержкой Java ME при ограниченных ресурсах). Другой вариант – клиент как набор html-страниц web-сервера – формально также является тонким. Для эффективной реализации необходимо подобрать технологию, максимально сохраняющую преемственность с текущей архитектурой и структурой кода. В качестве такой технологии можно использовать Ajax [58].

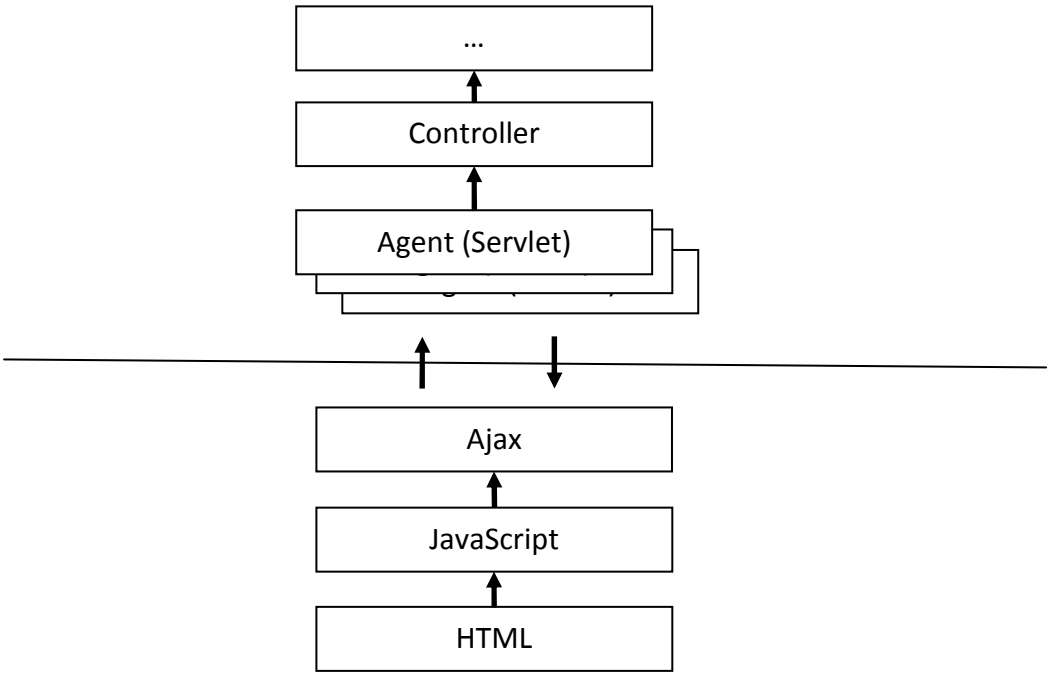

Рис. 5.13. Технология Ajax в тонком web-клиенте

Ajax позволяется посылать отдельные запросы к серверу, не привязанные к web-странице. Клиентская компонента протокола посылает собственный запрос к серверу, указывая необходимый активный ресурс (например, сервлет). Активный ресурс возвращает ответ в формате XML, из которого клиентская компонента извлекает необходимые данные. Применительно к описываемой архитектуре этот процесс будет выглядеть так:

- существующая разметка формы переносится как разметка web-страницы (HTML);
- на активные элементы страницы (кнопки, поля со списком), навешиваются события, обработчики которых реализуются на JavaScript;
- каждый обработчик посылает Ajax-запрос к сервлету, в котором передаются код команды и параметры;
- сервлет по коду команды выполняет соответствующее обращение к контроллеру на стороне сервера, полученные ответные данные передаются ответным сообщением в формате XML;
- поскольку контроллер запоминает текущее состояние представления, то на стороне сервера необходимо использовать механизм сессий для сохранения контекста соединения [59].

Примечание: в оригинале интерфейс представление-контроллер является двунаправленным, т.е. ответы контроллера оформлены как **асинхронные.** На самом деле ответы контроллер может посылать только после соответствующего запроса, но их может быть **несколько.** Чтобы вписать соответствующую схему в полностью синхронный протокол Ajax, контроллеру необходимо в интерфейс ViewControllerListener посылать еще одну команду – «окончание ответа». Тогда сервлет будет накапливать в ответы и посылать их единым сообщением.