Архитектурное проектирование

Архитектурное проектирование в наибольшей степени представляет «штучный продукт», основывается на личном опыте и кругозоре разработчиков. Архитектура включает в себя:

- **значимые решения** по поводу организации ПС
- **структурные элементы и их интерфейсы**, при помощи которых компонуется система
- **поведение** - взаимодействие между этими элементами
- **компоновка элементов** в иерархию подсистем
- **стиль архитектуры** который направляет эту организацию

Архитектурное описание является **многомерным,** т.к. описывает ПС с различных точек зрения и **минимально избыточной** (добавление данных не вносит качественных изменений в описание ПС, удаление ведет к потере существенных качеств).

Архитектурными сущностями являются элементы процесса проектирования (согласно своду знаний по программной инженерии SWEBOK):

- ключевые идеи и концепции: абстрагирование, связность и соединение, модульность и декомпозиция, инкапсуляция, разделение интерфейса и реализации;
- параллелизм;
- контроль и обработка событий;
- распределение компонентов;
- обработка ошибок и исключительных ситуаций и обеспечение отказоустойчивости;
- взаимодействие и представление (Model-View-Controller - MVC);
- сохраняемость данных (доступность «долгоживущих» данных);
- виды представления архитектуры - структурное, поведенческое, логическое, физическое, реализация кода;
- архитектурные стили;
- шаблоны проектирования;
- методы проектирования: нисходящее проектирование, модульное, абстрагирование, итеративность, функционально-ориентированное или структурное проектирование, объектно-ориентированное проектирование, проектирование на основе структур данных, компонентное проектирование.

Ключевые идеи

Все варианты приложений разрабатываются на основе системы функционально-ориентированных слоев:

- представление (View);
- поведение (Controller);
- бизнес-модель, бизнес-объекты (Model);
- табличные объекты БД (Data Access Objects - DAO);
- коннектор и библиотека доступа к СУБД;
- сервер БД.

Возможность реализации толстого и тонкого клиента на основе базового класса бизнес-модели, имеющего 3 реализации интерфейса получения бизнес-объектов:

- конструирование бизнес-объектов из табличных объектов DAO (толстый клиент)
- XML-сериализатор для Web API - (тонкий клиент)
- сериализатор в двоичные файлы (для локальных копий рейтинга)

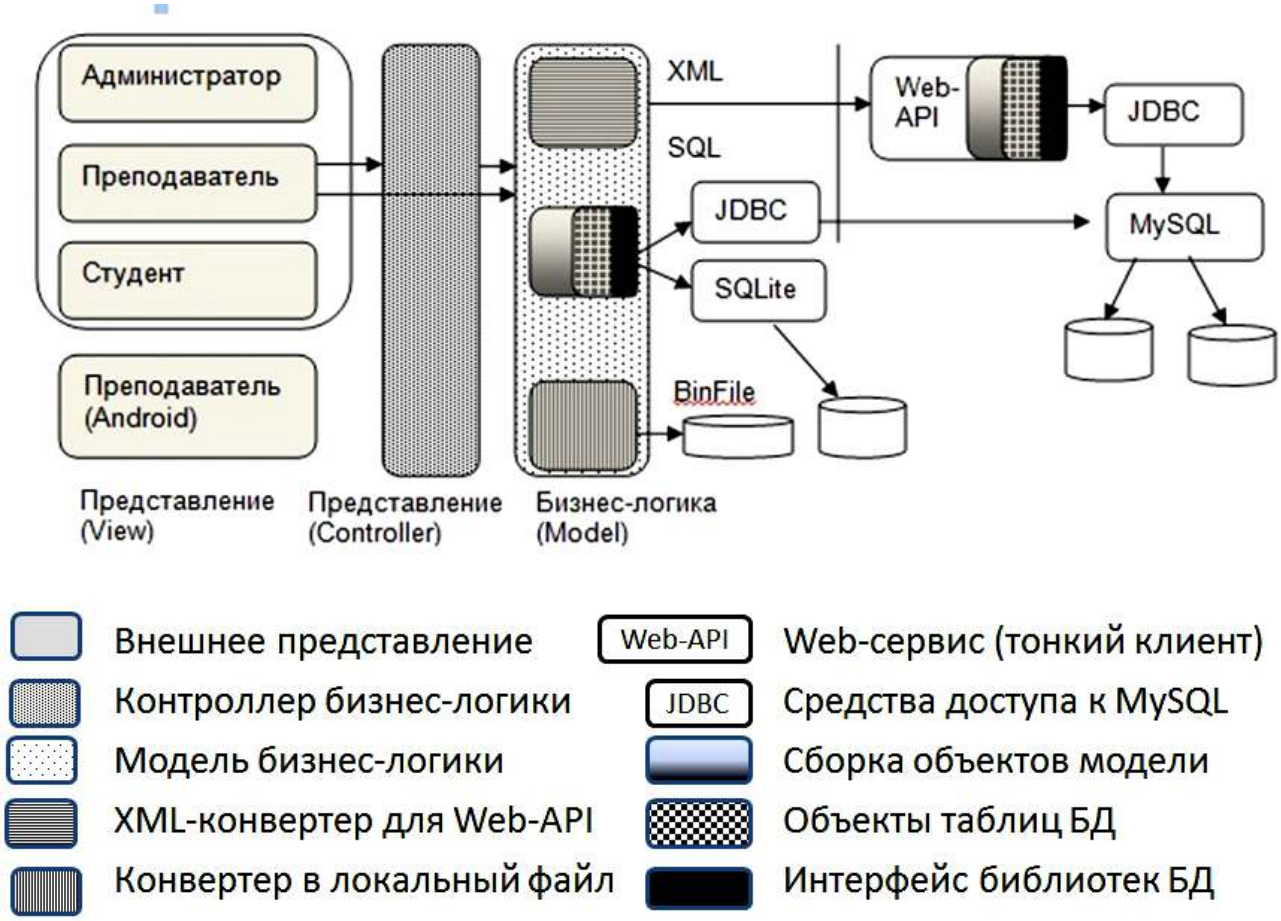

Рис. 5.3. Архитектура системы учета рейтинга успеваемости

Основными архитектурными компонентами являются:

- уровень представлений – реализация внешних форм клиентских приложений;
- контроллер бизнес логики – алгоритмы поведения для всех клиентов приложения “преподаватель”;
- модель бизнес-логики (классы бизнес-объектов) – внутреннее представление рейтингов, групп, студентов, оценок и т.п. и их основных алгоритмов;
- средства сборки бизнес-объектов из табличных объектов БД;
- конвертеры бизнес-объектов в двоичные файлы и в XML-формат;
- табличные объекты БД (DAO – Data Access Objects);
- интерфейс к библиотекам JDBC и SQLite;
- серверные компоненты web-сервиса (сервлеты).

Параллелизм. Синхронизация

Специальных требований к параллелизму не предъявляется. В представлениях (View) исполнение продолжительных операций должно проходить в фоновом режиме, при этом элементы управления не должны быть блокированы, но и не позволяют выполнять действия над данными, обрабатываемыми в фоновом режиме (“мягкая” блокировка GIU)

Взаимодействие и представление (MVC)

Все приложения используют уровни бизнес-объектов и ниже, т.е. общую модель представления данных. Все приложения, кроме “преподаватель” совмещают функции View+Controller в соответствующих классах, т.к. их поведение является уникальным. Приложение “преподаватель” имеет **полностью разделенные** компоненты View и Controller по причинам:

- имеется две версии приложения с идентичным поведением - desktop и мобильное (Android);
- при отсутствии соединения с БД приложения должны использовать источники данных - локальные копии рейтингов. Этот функционал прозрачен для уровня представлений и реализован в контроллере. Последний управляет переключением и синхронизацией источников данных.

Архитектурное проектирование и прикладные протоколы

Прикладной протокол, являясь связующим элементом архитектурных компонент, должен соответствовать требованиям, которые предъявляются к их взаимодействию. Поэтому те вопросы, которые необходимо задавать по структуре протокола, необходимо также продублировать на архитектурном уровне. Все они уже обсуждались, теперь соберем их воедино:

1. какие компоненты (слои, подсистемы) связывает протокол;
2. какие элементы функционала определяют форматы передаваемых сообщений протокола (классы предметной области, бизнес-объекты);
3. какие локальные интерфейсы «расшиваются» для удаленного доступа через данный протокол, требуется ли прозрачная реализация этих интерфейсов;
4. какие средства синхронизации данных используют связываемые компоненты, изменяются данные прикладного уровня одним или обоими участниками взаимодействия. Какие примитивы передачи данных используются (получение данных по запросу, контроль и передача обновлений и т.д.);
5. в какой из компонент происходят события, инициирующие взаимодействие, кто преимущественно играет роль клиента;
6. необходимо однонаправленное или двунаправленное, синхронное или асинхронное взаимодействие;
7. какой стандартный вид «линейного» драйвера передачи сообщений технологически лучше всего подходит для реализации протокола (сокеты, http-соединение, web-сокет);
8. необходимо ли создавать собственные средства повышения надежности передачи, контроля состояния (keep-alive) и восстановления соединений;
9. взаимодействие происходит в виде постоянных соединений, восстанавливаемых соединений или транзакций (без сохранения контекста),

10. в каком виде осуществляется поддержка контекста соединения (объекты, привязанные к соединению, внешняя поддержка сессий);
11. необходимо ли включать в протокол сообщения о сбоях (исключениях), происходящих при обработке запросов на прикладном уровне противоположной стороне, «транслировать» исключения и т.п.;
12. необходимы ли средства трассировки протокола и сбора статистики при его отладке и эксплуатации;
13. какие форматы сообщений использует протокол (собственные форматы, двоичная, XML- или JSON-сериализация объектов-сообщений);
14. какие элементы параллелизма необходимы в реализации протокола (потоки, синхронизация, буферизация, планирование);
15. предполагает ли протокол открытое использование (например, в виде webAPI), в каком виде необходимы его описания и спецификации;
16. какие стандартные технологии будут применены в его реализации его компонент;
17. какие стандартные средства защиты данных нижележащих уровней будут использованы и требуются ли собственные средства защиты.

Как видим, на большинство вопросов нельзя ответить, не имея полного представления о принятых архитектурных решениях и общих требований к разработке.