ЖИЗНЕННЫЙ ЦИКЛ ПРОЕКТА. ФРАГМЕНТЫ.

Анализ предметной области

ВУЗ территориально располагается в виде **учебных корпусов,** в каждом из которых имеется одна (как правило) или несколько **вахт.** На вахтах организовано дежурство **вахтеров,** обычно в виде суточных **смен.** Графики дежурств вахтеров более-менее постоянны, хотя возможны подмена и перевод из корпуса в корпус.

На вахте находятся ключи от **аудиторий,** аудитория имеет основной и **запасной** комплект ключей. На каждой вахте ведутся **журнал выдачи ключей** и **журнал постановки на пультовую охрану (постановка ни пульт).** В журнале выдачи ключей на **каждую дату** открывается новый блок записей, содержащих **время выдачи, номер аудитории, фамилию сотрудника и подпись** взявшего**, время сдачи и** подпись **сдавшего.** Журнал постановки на пульт заполняется в виде пустых записей на **каждую дату** под каждую аудиторию, для которой заполняется **время выдачи, номер аудитории, фамилия сотрудника и подпись** взявшего**, время сдачи, фамилия сотрудника и подпись** сдавшего. Если ключ берется и сдается несколько раз, то в журнале постановки на пульт делаются добавочные записи.

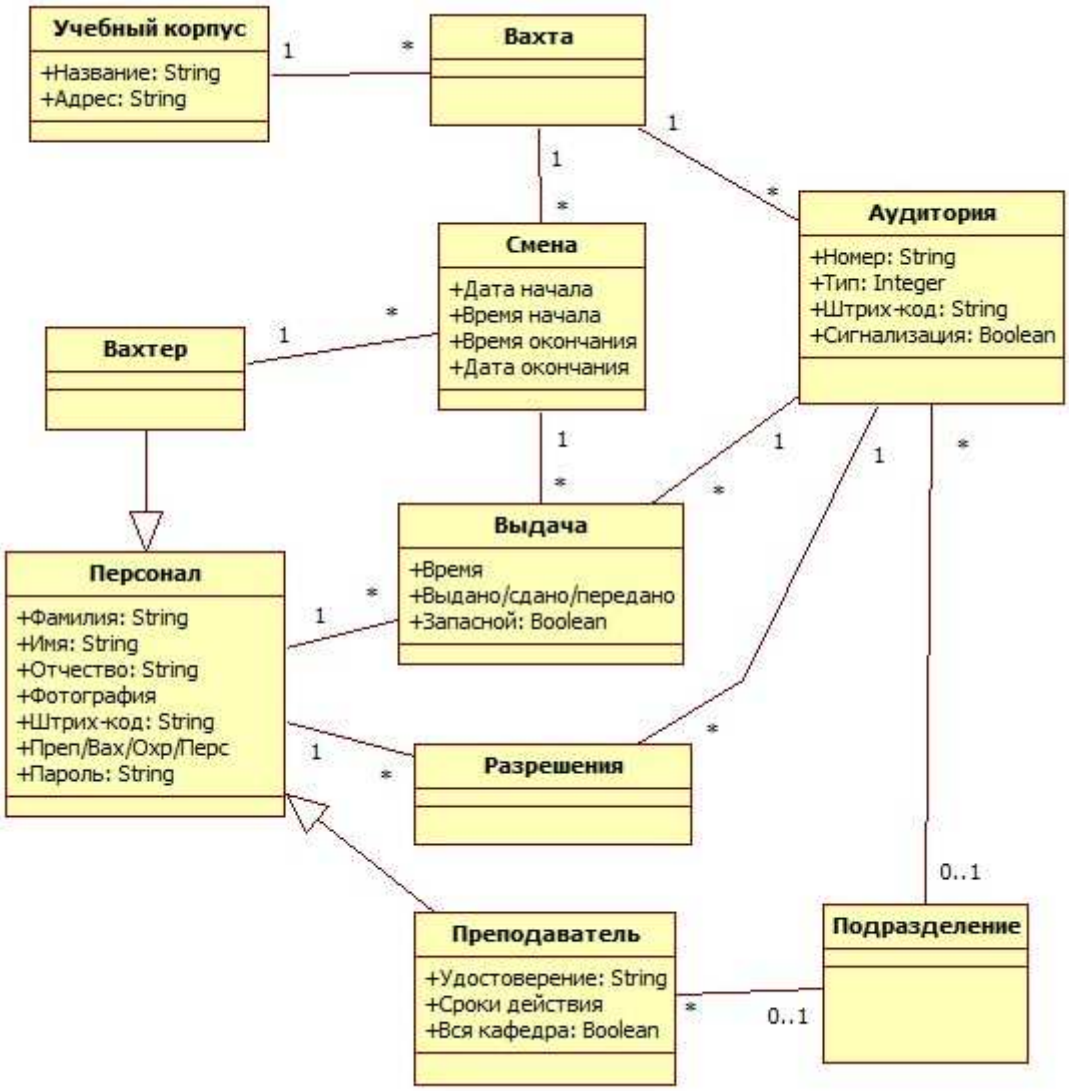
Кроме вахтеров, к системе имеют отношение **преподаватели,** ведущие учебный процесс, **персонал,** обслуживающий здание и учебный процесс (технички, электрики, буфет), а также **сотрудники отдела охраны,** в том числе **начальник.**

Преподаватель имеет **удостоверение**, в котором есть фамилия, имя и отчество, подразделение, фотография, сроки действия (продления) и номера аудиторий,

Аудитории могут подразделяться на **лаборатории и мультимедийные аудитории**, для которых разрешение должно быть явно прописано в удостоверении, **учебные аудитории**, доступ в которые разрешен все преподавателям, а также **служебные помещения,** в которые доступ разрешен **персоналу.**

Модель классов предметной области

Модель классов предметной области отражает существующее status quo.



Видение проекта

В настоящее время журналы ведутся не слишком аккуратно ввиду большого количества записей, ключи от аудиторий на переменах передаются от преподавателя к преподавателю не всегда с фиксацией этого факта в журнале, записи делаются неразборчиво. В то же время, полное следование регламенту порождает очереди. Необходима автоматизация, максимально сохраняющая существующие бизнес-процессы, учитывающая квалификацию и консерватизм пользователей с минимальным набором действий со стороны основных пользователей (вахтеров).

Анализ требований

Цитируем [1]. Выясняем ЧТО должна делать система и строим ее концептуальную модель. *Требование (requirement)* - желательное свойство, характеристика или условие, которым должна удовлетворять система в процессе своей эксплуатации. Применительно к ПС используется следующая *классификация требований*, которая получила название модели *FURPS+*, что соответствует первым буквам соответствующих категорий требований на английском языке:

- функциональные требования - *(Functionality)*
- требования удобства использования *(Usability)*
- требования надежности (Reliability)
- требования производительности *(Performance)*
- требования возможности сопровождения *(Supportability)*
- символом "+" обозначены дополнительные условия, к которым относятся:
    - проектные ограничения
    - требования управления системой
    - требования к графическому интерфейсу пользователя
    - физические требования
    - юридические требования

Функциональное требование - желаемое поведение системы с точки зрения ее пользователя. Функциональные требования реализуются функциями ПС.

Функция системы — поведение, которое необходимо реализовать в разрабатываемой программной системе. Если X действительно является функцией системы, то имеет смысл следующее предложение: система должна выполнять «X» Формулировка пожеланий (требований) заказчика может быть неконкретной, расплывчатой и не однозначной.

Каждое функциональное требование необходимо выявить, зафиксировать, уточнить, конкретизировать его смысл и поставить ему в соответствие одну или несколько функций системы. Некоторые требования могут быть абстрактными, т.е. им не будет соответствовать ни одна функция системы и они из дальнейшего рассмотрения исключаются.

Функциональные требования в RUP моделируются при помощи вариантов использования. **Вариант использования** (*Use Case, прецедент*) — внешняя спецификация последовательности действий, которые система может выполнять в процессе взаимодействия с действующими лицами (actor) **с целью получения значимого для них результата**.

Читая текст концепции (vision), фиксируем в функциональные требования, детализируем их в виде функций (бизнес-логики), каждую функцию отображаем соответствующим вариантом использования и сводим в трассировочную таблицу (матрицу).

Актер (actor), синонимы актант, действующее лицо — это абстрактное понятие, которое характеризует внешнего пользователя (или нескольких пользователей), взаимодействующего с системой. Каждый актер соответствует одной роли в которой выступает пользователь, взаимодействуя с системой. Каждый актер использует свои для него предназначенные сервисы, предоставляемые программной системой.

Особенности и требования к реализации

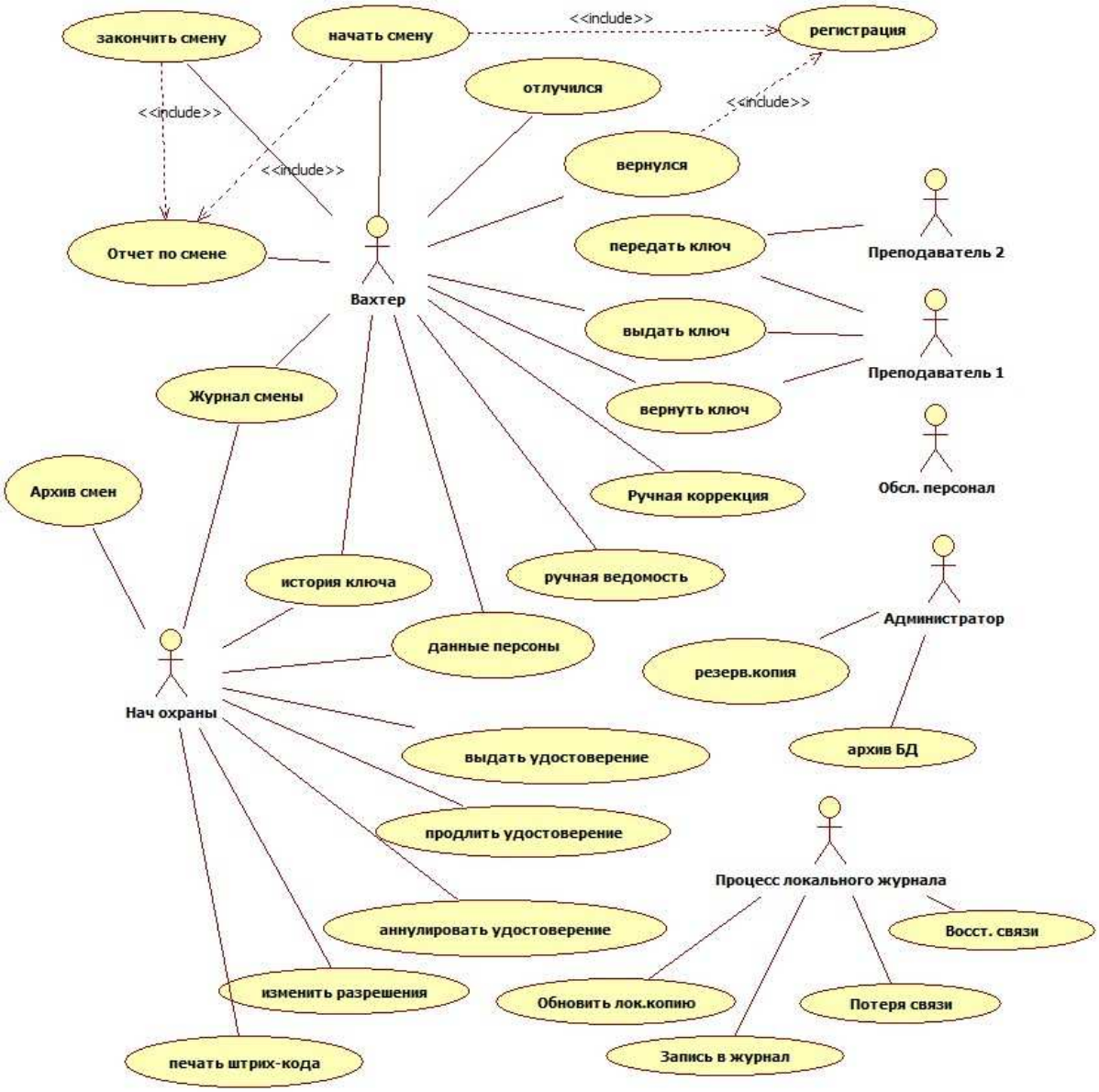
- Для вахтера система должна быть предельно простой, с минимальной последовательностью действий и использованием **привычных** устройств, которые встречаются в обыденной жизни. Необходим максимально возможный уровень подсказок;
- Необходимо использовать штрих-коды и оборудование для их считывания. С помощью штрих-кодов необходимо дублировать ввод **всех данных и команд,** т.е. штрих-коды должны быть уникальными во всех группах, например «KEY_7_211» или «ТСН_ВТ_РомановЕЛ». Штрих-кодами должны обладать:
    - Ключи аудиторий, или лучше ячейки, где они должны находиться, это надежнее и позволит, например, идентифицировать ключ при его отсутствии (например, при передаче);
    - Удостоверения преподавателей;
    - Бейджики персонала;
    - Бейджики вахтеров;
    - Доска меню основных команд, которые вахтер может использовать для работы с системой. **Штрих-коды команд можно выводить прямо на экран монитора**;
    - Сама вахта.
- Сканер кодов желательно использовать беспроводной, т.е. перемещаться с ним необходимо на расстояние до нескольких метров;
- Факты выдачи/приема ключей должны документироваться и подтверждаться подписью. Вместо журнала удобно использовать:
    - чековый принтер (или любой узкий принтер), с документом (чеком), на который преподаватель ставит подпись;
    - обычный принтер, в который вставляется узкая лента**;**
    - какой-либо иной способ обеспечения **доверия** к факту выдачи ключа.
- Бизнес-процессы автоматизированной системы должны быть максимально схожи с прототипом (в угоду консерватизму).
- К вопросу безопасности. При вводе данных со штрих-кодов пароли не запрашиваются. Контроль осуществляется вахтером, сканирование штрих-кода персонала сопровождается выводом фотографии. Сам вахтер при регистрации в начале смены и повторном входе вводит пароль. При отсутствии активности в течение некоторого времени система может закрываться (требовать повторной регистрации).
- Разрешение проблемных ситуаций. Система должна не сообщать об ошибках, а приводить себя в состояние, соответствующее действительности (исправлять ситуацию, добавлять события). Например, если ключ физически присутствует, но в БД отмечен как выданный, то надо ввести **фиктивную сдачу** в текущее время (возможно, вахтер не отметил факт сдачи).
- Надежность. Система должна обеспечивать ограниченную функциональность в различных нештатных ситуациях:
    - при пропадании (отсутствии) связи – накапливать данные при выполнении основного функционала и при восстановлении – сбрасывать в БД сервера, т.е. требуется иметь локальную копию части БД, касающейся текущей

вахты, частота ее обновления, например, а начале смены;

- при отключении электропитания – на резервном питании вывести документы ручного режима – ведомость выдачи с принятыми для ручного режима форматами, при восстановлении питания – привести БД системы в соответствии с ведомостью (или текущим состоянием), а ведомость оставить как печатный документ.

Актеры (роли) и прецеденты

Роли имеют все пользователи системы. В качестве псевдо-ролей выступают внутренние процессы, управляемые внешними событиями, таймерами и другими прецедентами, например, ведение журнала системы.



Архитектура

Цитируем RUP. Архитектура - это:

- Общая организация программной системы;
- Структура – структурные элементы, интерфейсы и связи, интерфейсы как элементы поведения (функционала);
- Иерархия и модульность;
- Архитектурный стиль.

Архитектура – минимально возможное описание программной системы, достаточное для понимания ее сущности и функционирования (**удаление с потерей целостности, добавление избыточно).**

Архитектура имеет **множественное представление** (срезы).

Представление	Роль	Содержание
Прецедентное	**Системный архитектор**	**Ключевые сценарии (прецеденты), описывающие реализацию функционала системы**
Логическое	Пользователь	Представление системы с точки зрения конечного пользователя (внешний вид, логическая структура).
Реализационное	Программист	Структура программного кода: пакеты, классы, библиотеки, интерфейсы, абстракции.
Процедурное	Системные интеграторы	Процессы, потоки, параллелизм, взаимодействие. Производительность, масштабируемость.
Представление распространения (развертывания)		Топология развертывания системы, распределение компонент по узлам сети, процедуры развертывания и администрирования

Прецедентное представление является **центральным –** рассмотрено выше**.**

Архитектурные стили:

- Многослойная структура толстый клиент – сервер. Синхронизация пользователей по данным – минимальная. Слои: представление – контроллер (возможно только для приложения «вахтер») - бизнес модель – ОО доступ к данным – библиотека доступа к БД (JDBC) – сервер БД;
- Представление – контроллер – модель.

Представление развертывания

На данном уровне не нуждается в проработке. Предполагает узел-сервер БД и некоторое количество клиентов со стандартным доступом через сеть.

Процедурное представление

На данном уровне не нуждается в проработке. Синхронизация параллельного доступа к БД минимальна из-за работы пользователей с независимыми данными. Количество пользователей ограничено, внутренний параллелизм приложения (работа с сетью, тайм-ауты) решается на этапе проектирования.

Логическое представление

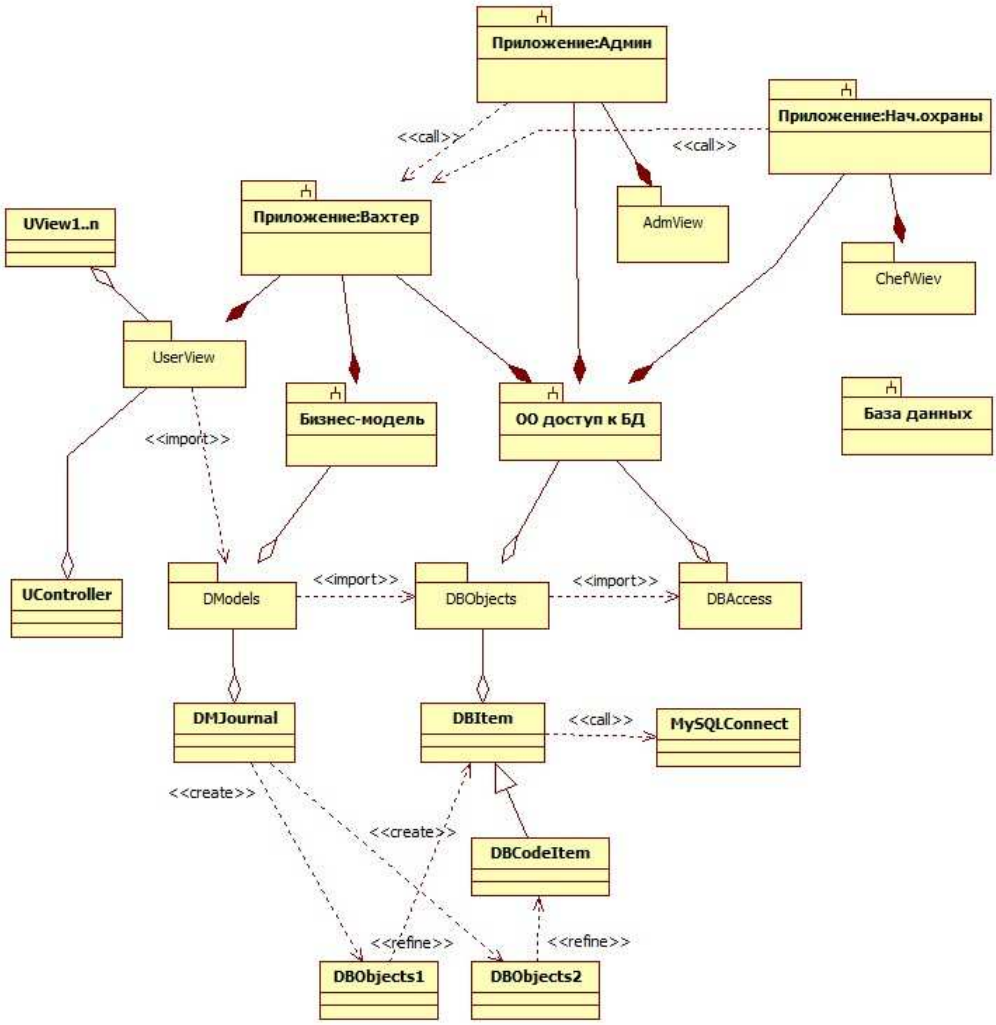
Прототип интерфейса пользователя. Выбор стиля интерфейса: «кабина самолета», табличная, мини-диалог. С учетом квалификации пользователей нужно использовать максимальное число простых форм-диалогов с развитой системой подсказок. Для документирования лучше всего – **диаграмма форм (экранов),** с указанием переходов, вводимых данных, подсказок (строится на основе прецедентов).

Реализационное представление

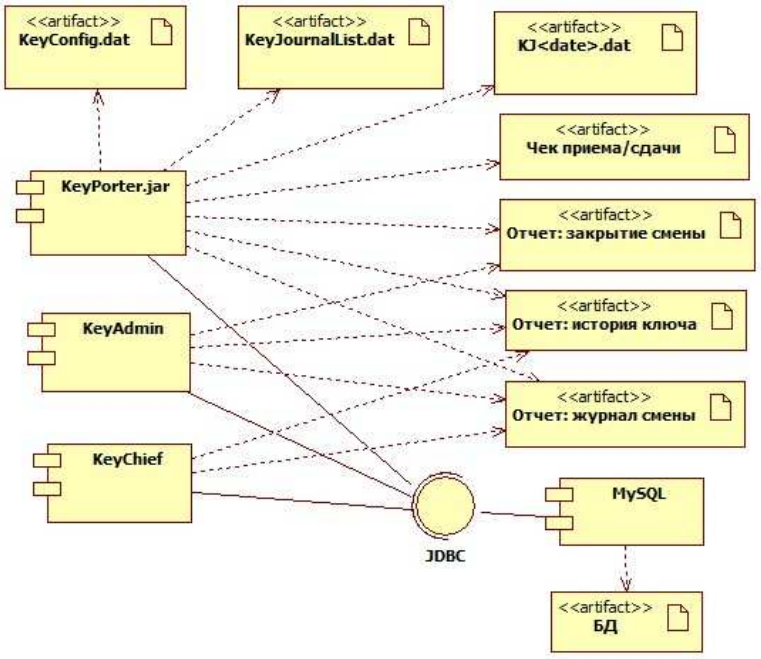
Самое сложное, т.к. касается «внутренних проблем», не отраженных в прецедентах. Делается на основе анализа требований (в основном, нефункциональных). Вопрос **«как»** наиболее уязвим для формального проектирования.

Вкратце сформулируем принятое архитектурное решение:

- Ввиду слабого взаимодействия пользователей по данным можно использовать стандартный дистанционный доступ к БД через драйвер JDBC в режиме толстого клиента. Пакет доступа данных обеспечивает необходимый сервис на уровне произвольных SQL-запросов;
- Следующий слой – объекты таблиц БД – строится на основе базового класса **DBItem**, обеспечивающего связку «производный класс – таблица БД – запись двоичного файла», что позволяет работать с произвольной физической моделью путем написания классов-наследников практически без функционала. Т.к. большинство сущностей имеют имена и штрих-коды, то следует упомянуть еще один базовый класс **DBCodeItem;**
- Слой бизнес-модели «собирает» из объектов физической модели необходимые логические представления. Это – объекты, необходимые для составления отчетов. Единственный важный класс на этом уровне – логический объект «смена», интегрирующий все данные, необходимые для текущей смены, выбранные из БД (например, **персонал, подразделения, аудитории, разрешения, выдача).** Эти данные образуют объект класса «журнал», который обеспечивает автономную работу в проблемных ситуациях и синхронизируется с основной БД. Объекты-журналы сохраняются в локальных файлах;
- Слой внешнего представления. При наличии сложной диаграммы форм (экранов) лучше всего интегрировать их через контроллер, обеспечивающий нужную последовательность состояний и событий для классов-представлений.



Вспомогательным элементом может быть **диаграмма компонентов,** включающая программные компоненты, БД и файлы (данных, параметров конфигурации и отчеты).

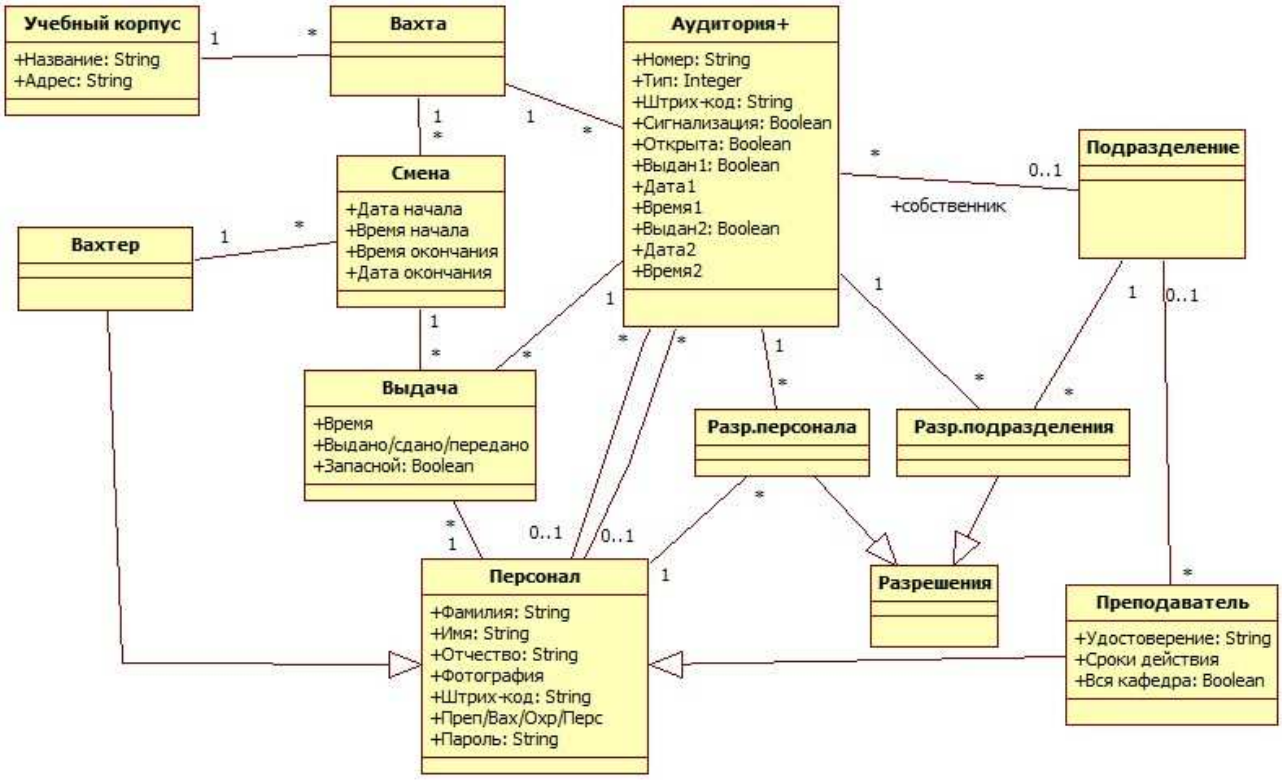


Модель классов анализа

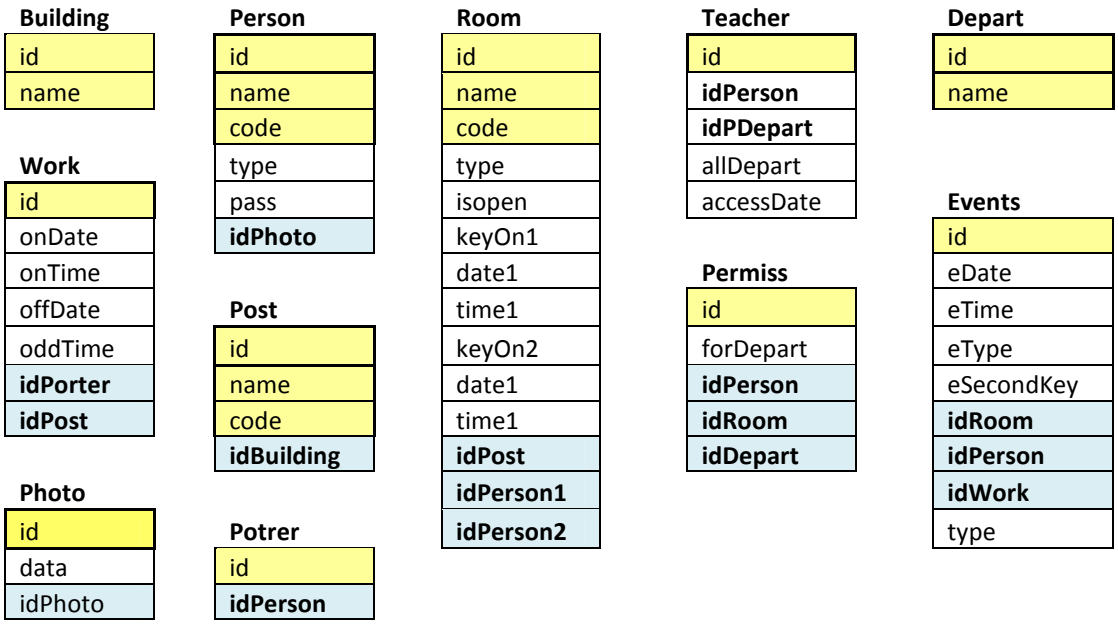
Модель классов анализа развивает модель классов предметной области, но учитывает требования со стороны требований, прецедентов и архитектуры. В нашей модели появляются следующие изменения:

- требования: выдачу разрешений удобно производить не только по отдельным преподавателям, но и для подразделения в целом. В модели появляется базовый класс разрешений и два производных класса – разрешения **для персонала** и разрешения **для подразделений**;
- реализация прецедентов: для установления состояния аудитории (открыта/закрыта) необходимо производить поиск связанного объекта с последней датой/временем классе «выдача». К тому же возможны разные нестандартные ситуации, когда ключ сдан, а аудитория открыта, либо наоборот – ключ не сдан, но аудитория закрыта. Поэтому в класс «аудитория» необходимо продублировать данные о последней выдаче для основного и запасного ключей. Тогда класс «выдача» становится **чистым журналом событий,** доступ к которому необязателен для ведения текущих операций;

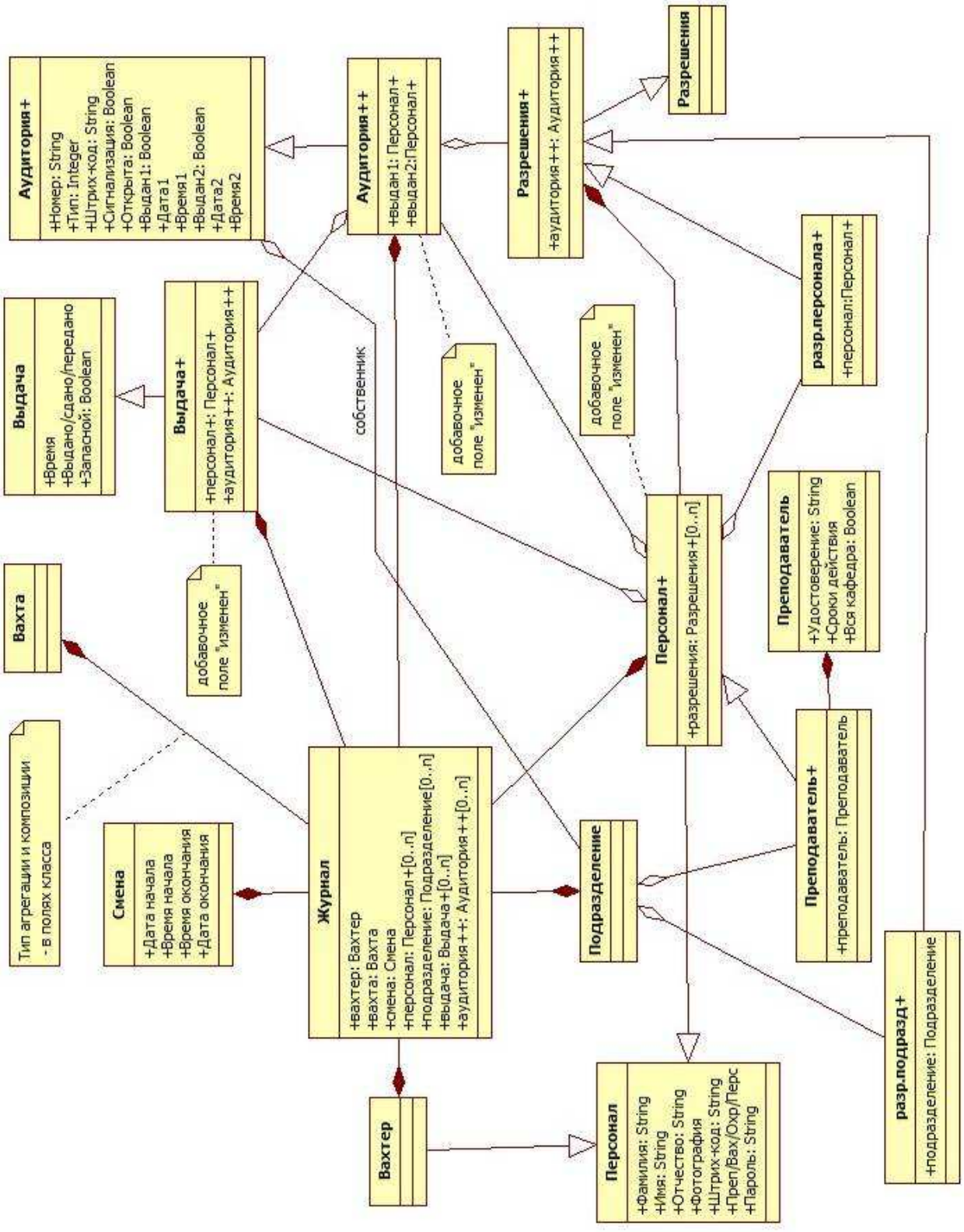
  С учетом этих изменений модель классов анализа выглядит несколько иначе:



В разные подсистемы модель проецируется особым образом. В модели БД объекты классов-таблиц БД имеют уникальные идентификаторы и реализуют связи в виде целочисленных идентификаторов связанных объектов. Эти классы могут быть получены напрямую из модели анализа. Аналогичный вид имеет структура БД.



При проектировании бизнес-модели создается своя структура, состоящая из иерархической композиции объектов - «журнал». «Журнал» содержит локальную копию данных модели анализа для текущей смены, может обновляться автономно (при пропадании связи с БД) и синхронизируется с последней. Ассоциации в модели анализа заменяются композициями (массив или вектор прямых ссылок) или агрегациями (поле – прямая ссылка).


Аудитория+
+Номер: String
+Тип: Integer
+Штрих-код: String
+Сигнализация: Boolean
+Открыта: Boolean
+Выдан1: Boolean
+Дата1
+Время1
+Выдан2: Boolean
+Дата2
+Время2
Аудитория++
+выдан1: Персонал+
+выдан2:Персонал+
Разрешения+
+аудитория++: Аудитория++
Разрешения
Выдача
+Время
+Выдано/сдано/передано
+Запасной: Boolean
Выдача+
+персонал+: Персонал+
+аудитория++: Аудитория++
собственник
добавочное поле "изменен"
добавочное поле "изменен"
разр.персонала+
+персонал:Персонал+
Вахта
добавочное поле "изменен"
Персонал+
+разрешения: Разрешения+[0..n]
Преподаватель
+Удостоверение: String
+Сроки действия
+Вся кафедра: Boolean
Тип агрегации и композиции
- в полях класса
Смена
+Дата начала
+Время начала
+Время окончания
+Дата окончания
Журнал
+вахтер: Вахтер
+вахта: Вахта
+смена: Смена
+персонал: Персонал+[0..n]
+подразделение: Подразделение[0..n]
+выдача: Выдача+[0..n]
+аудитория++: Аудитория++[0..n]
Подразделение
Преподаватель+
+преподаватель: Преподаватель
разр.подразд+
+подразделение: Подразделение
Вахтер
Персонал
+Фамилия: String
+Имя: String
+Отчество: String
+Фотография
+Штрих-код: String
+Преп/Вах/Охр/Перс
+Пароль: String

Соединение архитектуры с прецедентами. Диаграммы устойчивости и диаграммы последовательности

Цитируем [1]. Для базовых прецедентов создаются прототипы ЭФ и пишутся сценарии.

Сценарий — это текстовое описание потока событий при выполнении конкретного варианта использования, выражающее некий аспект поведения системы.

Сценарии служат для перехода от вариантов использования к объектам программной системы. Анализируются имена-существительные в тексте сценария. Некоторые из них будут действующими лицами, другие — объектами, а третьи — атрибутами объекта.

Пример сценария

Прототип пользовательского интерфейса.

Наименование прецедента. Выдать ключ.

Актеры. Вахтер. Преподаватель.

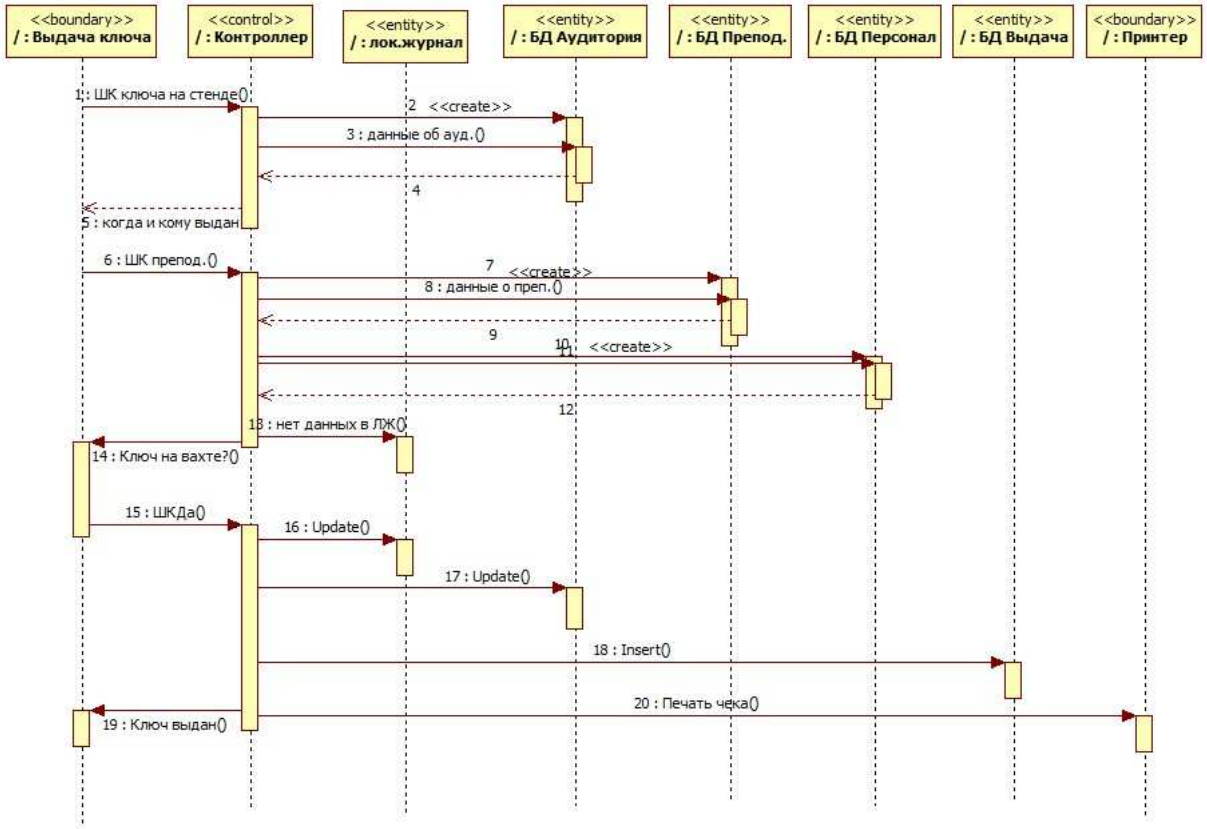
Предусловия. Вахтер зарегистрирован. Смена открыта. Открыт локальный журнал.

Цель исполнения.

Краткое описание.

Поток событий

1.Вахтер: Сканирует штрих-код ключа на стенде	Если есть соединение с БД, берет состояние ключа из БД, иначе из локального журнала.
	Если ключ уже выдан, поле со справкой – когда и кому.
2. Вахтер: Сканирует штрих-код преподавателя	Если есть соединение с БД, данные берутся оттуда, иначе – из локального журнала. Если в локальном журнале нет данных, по **преподаватель заносится в локальный журнал** без разрешений (по фамилии на штрих-коде).
3.	Если тип аудитории – лаборатория, то проверка разрешения. Нет разрешения – отказ (держится 10 сек.). Иначе выводится «Выдать ключ».
4. Вахтер: Сканирует штрих-код команды «выдать»	Если ключ отмечен, как выданный, выводится вопрос «Ключ на вахте?», иначе переход к п.7.
5. Вахтер: Сканирует штрих-код команды «да»	Оформляется фиктивный возврат ключа. Переход к п.7.
6. Вахтер: Сканирует штрих-код команды «нет»	Завершение сценария
7.	Если есть соединение с БД – редактируется «аудитория» и добавляется «выдача». Редактируется локальный журнал. Печатается чек.
8. Преподаватель подписывает чек.	



Ссылки

1. Практический_анализ_по_RUP_(Николай_Киреев,_AnalystDays-2012).