

# Лекция 11: Предметно-ориентированное проектирование, стратегические аспекты

Юрий Литвинов  
y.litvinov@spbu.ru

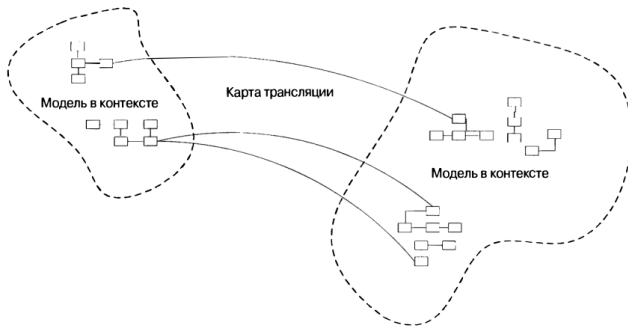
25.11.2024

# Проблемы DDD в больших системах

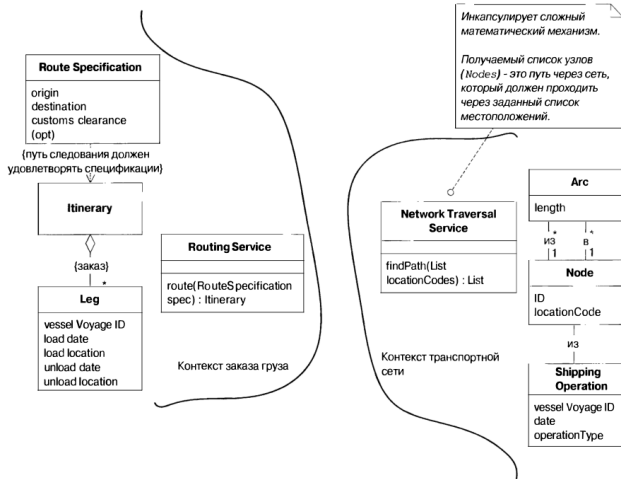
- ▶ Несколько команд => несколько видений продукта
- ▶ Модель предметной области
  - ▶ Интегрированная — слишком большие затраты на поддержание целостности, слишком общая модель, чтобы быть полезной
  - ▶ Фрагментированная — затрудняет переиспользование и интеграцию системы
- ▶ Опасность ошибок при интеграции и переиспользовании
  - ▶ Класс “Платёж” — платёж поставщику или платёж клиента

# Принципы поддержания целостности модели

- ▶ Ограниченный контекст (Bounded context)
- ▶ Непрерывная интеграция (Continuous Integration)
- ▶ Карта контекстов (Context map)



# Пример, границы контекстов

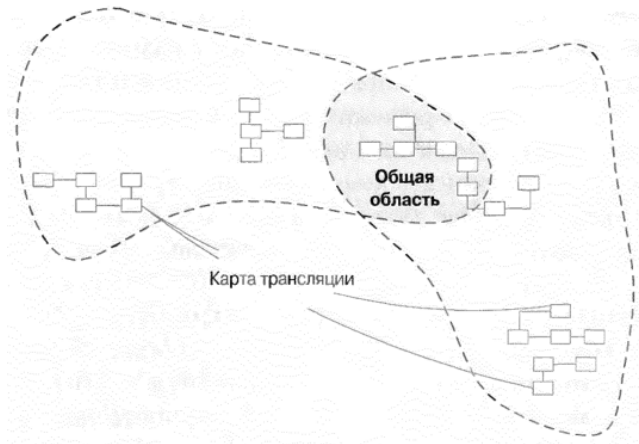


## Типовые ситуации интеграции контекстов

- ▶ Типовые ситуации интеграции контекстов и типовые пути действий в разных ситуациях
- ▶ Пути действий именованы и документированы (паттерны!)
- ▶ Чаще всего, выбор паттерна определяется обстоятельствами, не зависящими от команды
- ▶ Иногда отсутствие какой-либо интеграции — лучший способ интеграции

# Общее ядро

Shared Kernel



# Заказчик-поставщик

## Customer-Supplier

- ▶ Имеет смысл, когда одна компонента целиком зависит от другой
- ▶ Может привести к блокированию действий одной или другой команды
- ▶ Следует явно зафиксировать отношения между командами
  - ▶ Одна выступает в роли заказчика (одного из заказчиков) — участвует в планировании, поставяет задачи
  - ▶ Автоматизированные приёмочные тесты
- ▶ Желательно, чтобы команды находились в одной иерархии управления

# Конформист

## Conformist

- ▶ Имеет смысл, когда нет способа повлиять на компонент, от которого полностью зависим
  - ▶ Legacy-приложение, навязанная сверху технология и т.п.
- ▶ Просто принимаем модель и миропонимание “основного” компонента
- ▶ Не всегда плохо: чужой код может на самом деле выражать большее понимание предметной области



# Предохранительный уровень

## Anticorruption Layer

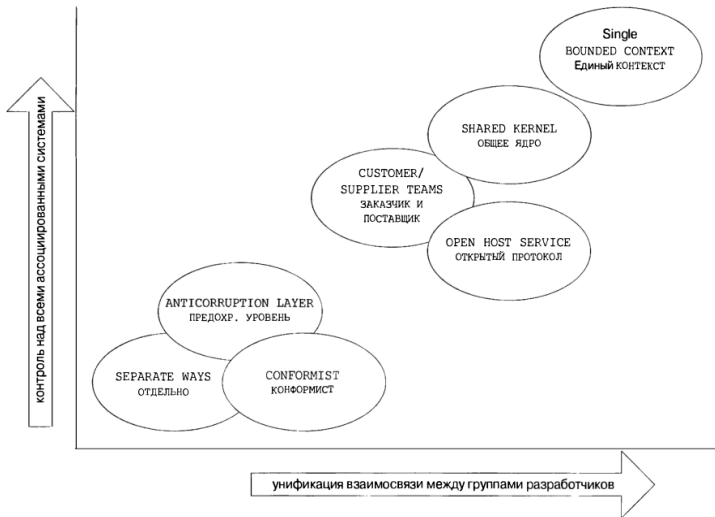
- ▶ Имеет смысл, когда “Конформист” не подходит
- ▶ Кусок кода (возможно, большой и страшный), отвечающий за трансляцию из одной модели в другую
  - ▶ Паттерны “Фасад” и “Адаптер”



## Ещё приёмы

- ▶ **Отдельное существование (Separate Ways)** — когда преимущества от интеграции меньше затрат на неё
- ▶ **Служба с открытым протоколом (Open Host Service)** — когда клиентов много
- ▶ **Общедоступный язык (Published Language)** — когда клиентов очень много, общая среда для общения

# Итого, шаблоны интеграции



# Пример: унификация слона

Шесть седовласых мудрецов  
Сошлись из разных стран.  
К несчастью, каждый был незряч,  
Зато умом блистал.  
Они исследовать слона  
Явились в Индостан.

Один погладил бок слона.  
Довольный тем сполна,  
Сказал он: "Истина теперь  
Как божий день видна:  
Предмет, что мы зовем слоном,  
Отвесная стена!"

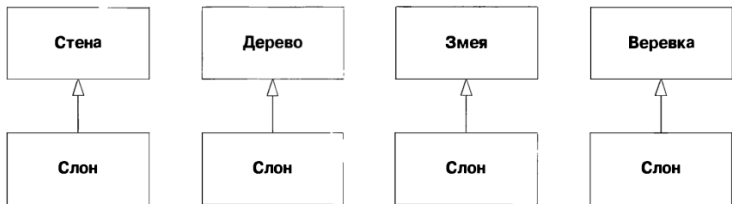
А третий хобот в руки взял  
И закричал: "Друзья!  
Гораздо проще наш вопрос,  
Уверен в этом я!  
Сей слон — живое существо,  
А именно змея!"

Мудрец четвертый обхватил  
Одну из ног слона  
И важно молвил: "Это ствол,  
Картина мне ясна!  
Слон — дерево, что зацветет,  
Когда придет весна!"

Тем временем шестой из них  
Добрался до хвоста.  
И рассмеялся от того,  
Как истина проста.  
"Ваш слон — веревка. Если ж нет  
Зашейте мне уста!"

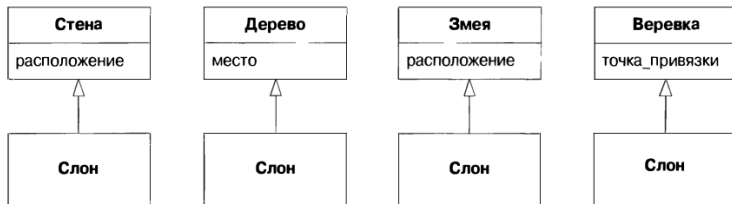
А как известно, мудрецам  
Присущ упрямый нрав.  
Спор развязав, они дошли  
Едва ль не до расправ.  
Но правды ни один не знал,  
Хотя был в чем-то прав.

# Унификация слона, Separate ways



# Слон, минимальная интеграция

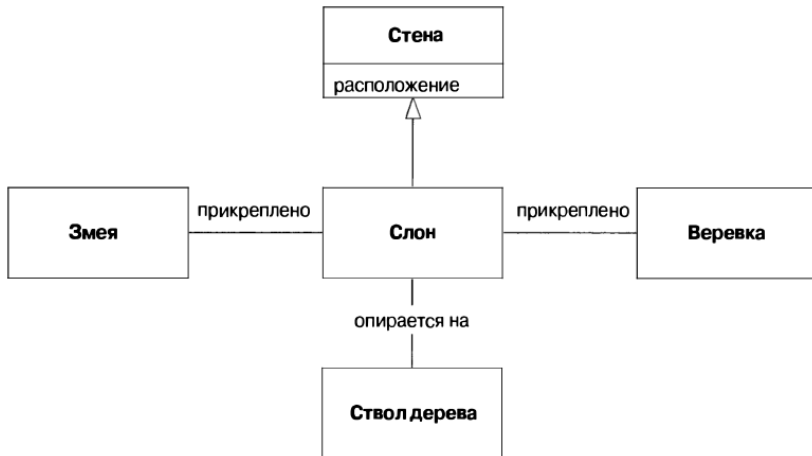
## Anticorruption Layer



**Трансляция:** {Стена.расположение ↔ Дерево.место ↔ Змея.расположение ↔ Веревка.точка\_привязки}

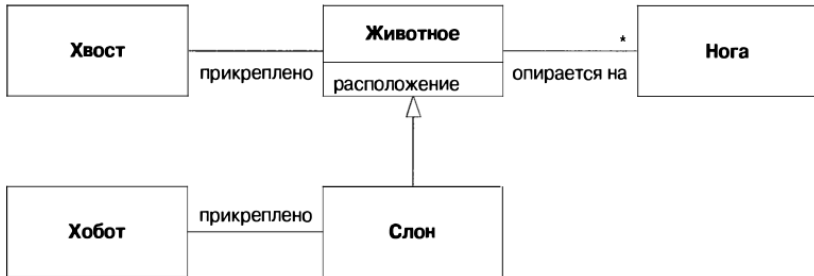
# Слон, слабая интеграция

Shared Kernel



# Слон, сильная интеграция

## Bounded Context





# Дистилляция

- ▶ **Дистилляция** — процесс выделения самого существенного в системе и отделения его от вспомогательного кода
- ▶ **Смысловое ядро (Core Domain)** — то, что, собственно, делает систему ценной
  - ▶ Должно быть минимальным и чётко отделённым от остальных компонентов системы
  - ▶ Опытные программисты не любят им заниматься, с этим надо бороться
  - ▶ Только Core Domain, фактически, составляет know-how

# Приёмы дистилляции

- ▶ **Domain Vision Statement** — документ (на одну страницу), описывающий смысловое ядро и его полезность
- ▶ **Выделенное ядро (Highlighted Core)**
  - ▶ Дистилляционный документ — 3-7 страниц текста про то, что составляет смысловое ядро и как его элементы взаимодействуют друг с другом
  - ▶ Flagged Core — элементы ядра выделены на существующей модели
- ▶ **Неспециализированные подобласти (Generic Subdomains)** — куски кода, неспецифичные для системы
- ▶ **Связный механизм (Cohesive Mechanism)** — куски кода, неспецифичные для предметной области вообще
  - ▶ Технические вещи, типа графов

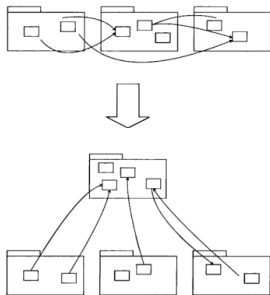
# Пример, грузоперевозки







# Абстрактное ядро



- ▶ Применяется, когда даже ядро оказывается слишком большим
- ▶ Состоит из абстрактных классов, которые потом реализуют отдельные модули

# Крупномасштабная структура

- ▶ **Крупномасштабная структура** — набор общих правил, по которым строится система или группа систем
- ▶ Должна эволюционировать вместе с моделью и кодом
- ▶ Не должна быть слишком жёсткой
  - ▶ Модель “Архитектор в башне из слоновой кости” не работает
- ▶ Лучше какая-то, чем никакой
- ▶ Небольшие проекты могут прекрасно жить и без всего этого
- ▶ Самая полезная структура — общий язык

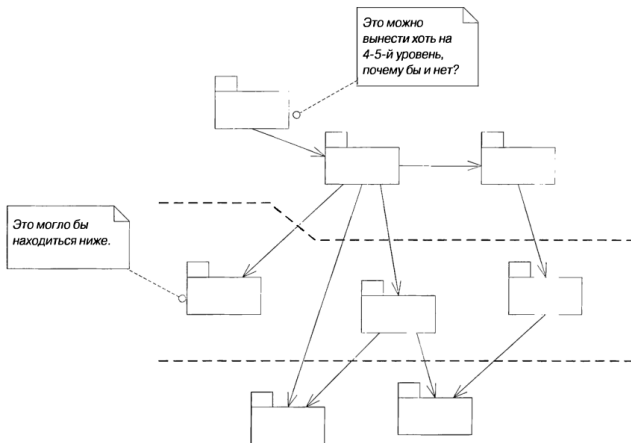
# Метафора системы

- ▶ **Метафора** определяет то, как в целом понимать систему
  - ▶ Множества примеров: рабочий стол, firewall и т.д.
- ▶ Метафора не всегда есть
- ▶ Метафора может быть опасной
  - ▶ Метафора тащит за собой лишний смысл



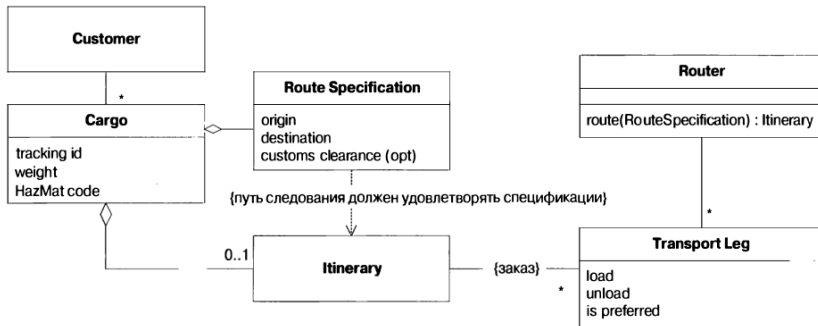
# Уровневая структура

Не должна быть механической

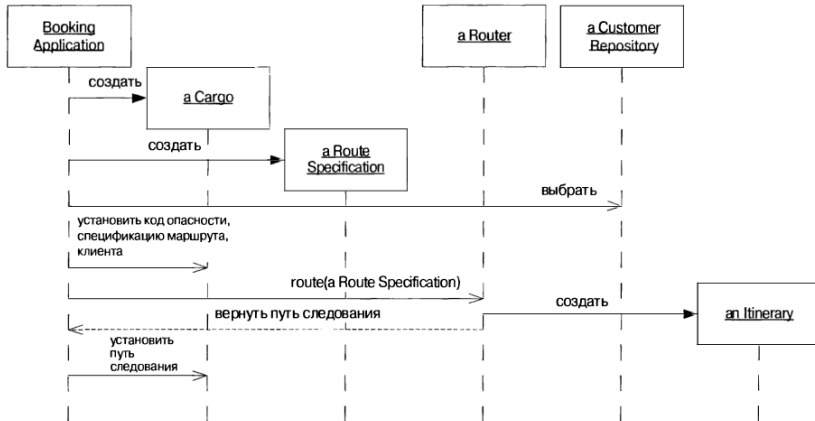


# Пример, перевозка грузов

## Исходная модель



# Установка пути следования

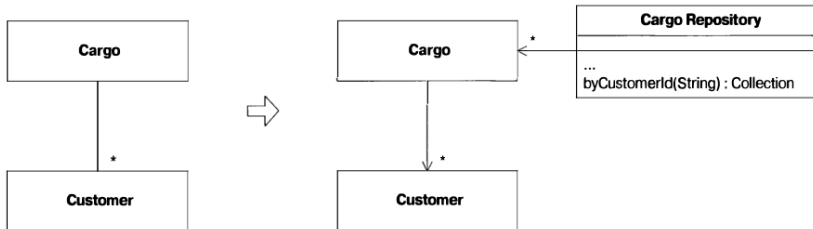


# Рефакторинг

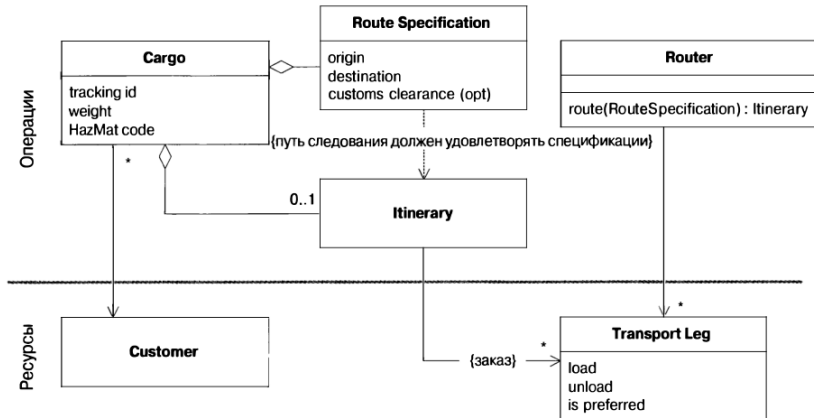
Два уровня:

- ▶ ресурсный — то, что обеспечивает наши возможности
- ▶ операционный — то, как мы пользуемся нашими возможностями

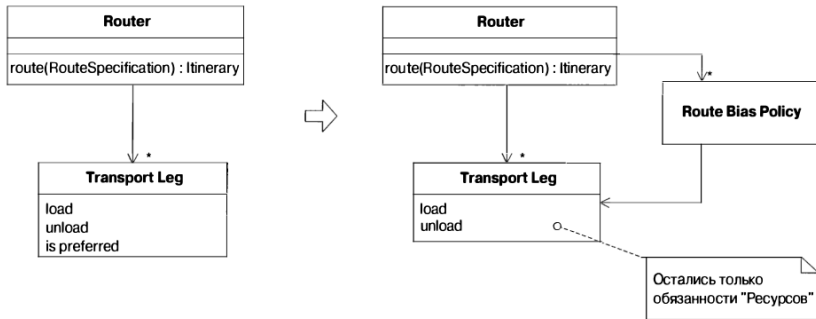
Двунаправленная связь между *Customer* и *Cargo* мешает



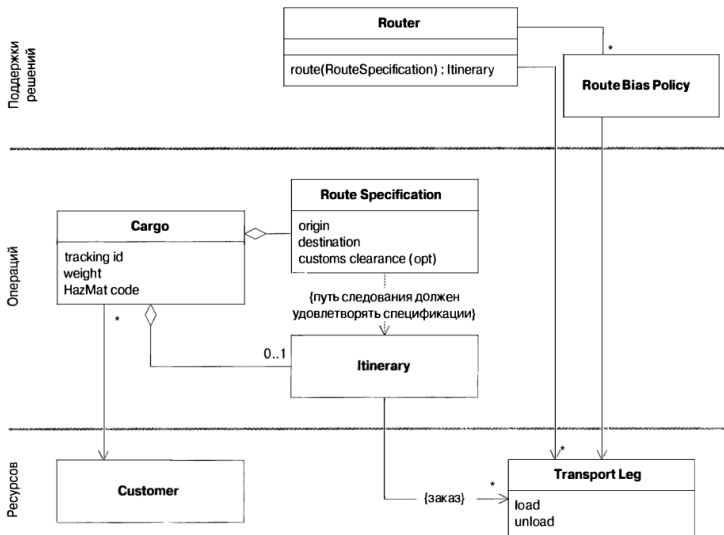
# Два уровня



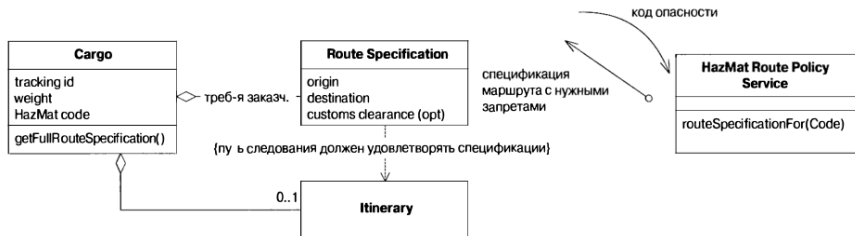
# Рефакторинг, выделение уровня принятия решений



## Три уровня



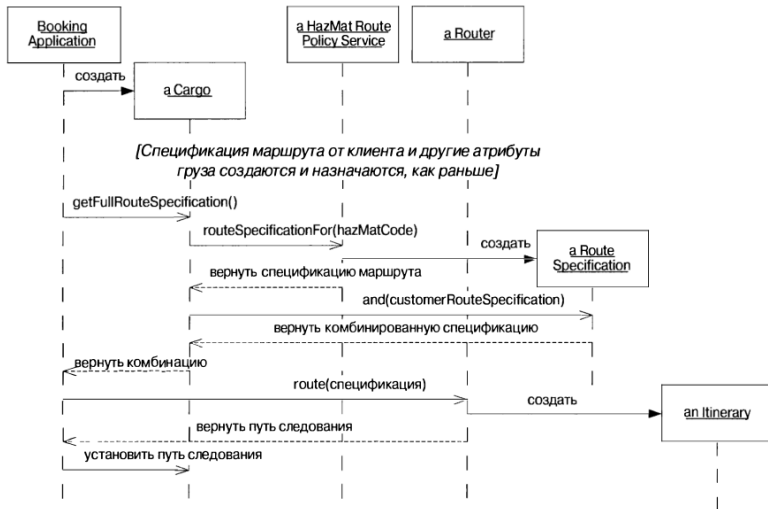
# Работа с опасными грузами, первая версия



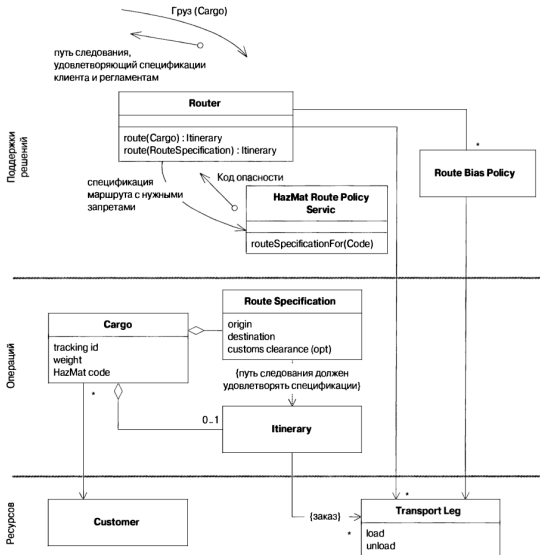


# Диаграмма последовательностей

## Работа с опасными грузами, первая версия

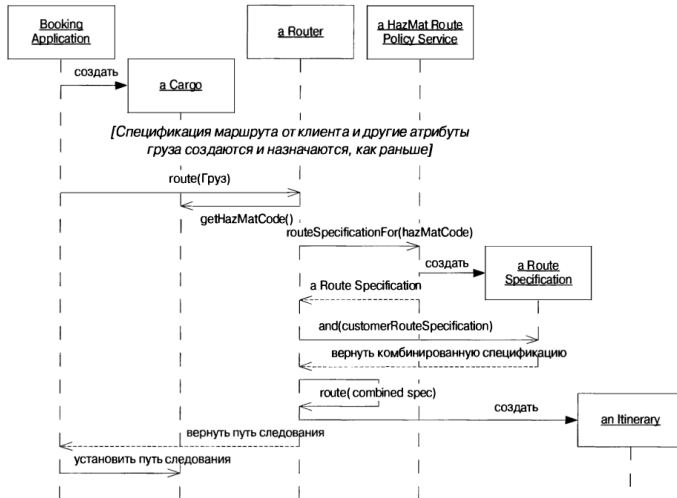


# Работа с опасными грузами, вторая версия



# Диаграмма последовательностей

## Работа с опасными грузами, вторая версия



# Типичные уровни в системах автоматизации производства

Принятия решений	Аналитические механизмы	Практически отсутствует как состояние, так и его изменение	Анализ управления Оптимизация использования Сокращение рабочего цикла ...
Регламентный	Стратегии Связи-ограничения (на основании целей или закономерностей данной отрасли)	Медленное изменение состояния	Приоритет изделий Предписанные регламенты изготовления деталей ..
Операционный	Состояние, отражающее реальное положение дел (деятельности и планов)	Быстрое изменение состояния	Инвентарная опись Учет состояния незавершенных деталей ...
Потенциальный	Состояние, отражающее реальное положение дел (ресурсов)	Изменение состояния в среднем темпе	Возможности оборудования Наличие оборудования Перемещение по территории ...

зависимость



# Типичные уровни в финансовых системах

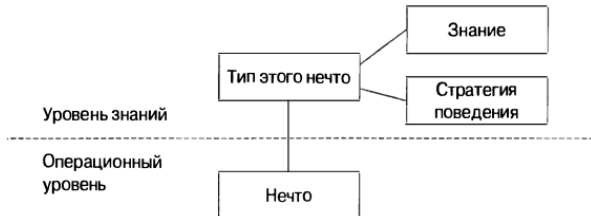
Принятия решений	Аналитические механизмы	Практически отсутствует как состояние, так и его изменение	Анализ рисков Анализ портфелей Средства ведения переговоров ...
Регламентный	Стратегии Связи-ограничения (на основании целей или закономерностей данной отрасли)	Медленное изменение состояния	Пределы резервов Цели размещения активов ...
Обязательств	Состояние, отражающее сделки и договоры с клиентами	Изменение состояния в среднем темпе	Соглашения с клиентами Соглашения по синдикации ...
Опера- ционный	Состояние, отражающее реальное положение дел (деятельности и планов)	Быстрое изменение состояния	Состояние кредитов Начисления Выплаты и распределения ...

зависимость



## Другие высокоуровневые структуры

- ▶ **Уровень знаний (Knowledge level)** использует информацию о типах сущностей, позволяя гибко переконфигурировать систему



- ▶ **Подключаемые компоненты (Pluggable Component Framework)** — стиль, описывающий общее ядро и набор взаимозаменяемых плагинов, которыми оно управляет
- ▶ Разные стили не исключают друг друга!