# О концептуальном моделировании

Б. Тальхайм (Германия)*

Технологии баз данных и информационных систем претерпели существенные изменения. В настоящее время повсеместное распространение получили системы управления контентом, веб-сервисы с интенсивной обработкой информации, сотрудничающие системы, интернет-базы данных, OLAP-базы данных, производящих анализ в реальном времени, и т. п. В то же время благодаря широкому применению технология объектно-реляционных данных стала стабильной и хорошо разработанной. Концептуальное моделирование (пока) не охватило перечисленные выше области. На протяжении десятилетий основным вопросом концептуального моделирования была спецификация структур, тогда как для адекватного представления современных систем необходимо также рассматривать вопросы функциональности, взаимодействия и распределенности. Многие задачи, поставленные еще в 1987 году в работах [13, 14] до сих пор остаются открытыми. В то же время появился целый ряд новых технологий, например объектно-реляционная модели и модели на основе XML. Новые технологии не решили всех проблем, скорее расширили и обострили имевшиеся нерешенные задачи. В работе приводится список открытых задач в классических областях теории баз данных — спецификациях структуры и функциональности. Задачи, связанные с взаимодействием и распределенностью, в настоящее время являются предметом весьма интенсивных исследований.

Постановки открытых задач сопровождается описанием основных результатов в области концептуального моделирования. Представляется подход к моделированию объектно-

*Bernhard Thalheim. *Computer Science and Applied Mathematics Institute, University Kiel, Olshausenstrasse 40, 24098 Kiel, Germany. Email: thalheim@is.informatik.uni-kiel.de*

реляционных сотрудничающих систем с поддержкой виртуальных рабочих групп, интеграции информационных систем, разнообразных архитектур, таких как OLTP-OLAP, play-out и play-in систем и систем анализа данных. Основой работы является расширенная модель отношения элементов (Entity-Relationship, ER), покрывающая все структурные возможности объектно-реляционных систем и использующая теории медиатипов и сценариев для спецификации взаимодействия.

# 1. Введение

## 1.1. Проектирование и разработка информационных систем

Проблема создания информационных систем может быть сформулирована следующим образом:
Для данной СУБД (или парадигмы базы данных) создать физическую и логическую структуру информационной системы так, чтобы система содержала все данные, необходимые для пользователей и для эффективного функционирования системы для всех пользователей. Определить прикладные процессы базы данных и их взаимодействие с пользователями.

Основными задачами создания баз данных являются:

- удовлетворение всех информационных (контекстных) требований для всех пользователей в данной прикладной области;
- реализация «естественной» и простой для понимания структуры хранимой информации;
- обеспечение готовности всей семантической информации для возможных изменений структуры;
- обеспечение требований к обработке данных и высокой эффективности обработки;
- обеспечение логической независимости запросов и транзакций на данном уровне;
- предоставление простых и понятных пользовательских интерфейсов.

В последние годы велось активное обсуждение структур баз данных. Некоторые проблемы были успешно решены. Однако при рассмотрении задачи моделирования возникают новые аспекты:

Структурирование приложений баз данных относительно структуры баз данных и соответствующих статических ограничений целостности.

Функционирование приложений баз данных на основе процессов и динамических ограничений целостности.

Распределение компонент информационной системы, задаваемое явной спецификацией сервисов и интерфейсов взаимодействия.

Интерактивность (взаимодействие с пользователями), определяемое на основе предполагаемых сценариев работы воображаемых пользователей, зависящих от типов носителей, используемых для выдачи информации пользователям и для обработки поступающих данных.

Понимание важности новых аспектов привело к созданию так называемого подхода к соразработке при моделировании на основе спецификаций **структуры, функциональности, распределения и интерактивности**. Каждый из аспектов имеет как синтаксические, так и семантические элементы.

Однако основную задачу решить так и не удалось.

**Открытая задача 1.** *Найти общую мотивацию, общую формальную модель и соответствие, удовлетворяющее всем свойствам, и формализовать характеристики.*

## 1.2. Общие модели информационных систем

Создание баз данных основывается на одной или нескольких моделях данных. Часто создание сводится исключительно к структурным аспектам. Иногда дополнительно вводится статическая семантика, основанная на статических ограничениях целостности. После реализации структуры происходит спецификация процессов. Поведение процессов специфицируется посредством динамических ограничений целостности. Далее выполняется разработка интерфейсов. Глубина

теоретической проработки описанных выше этапов существенно различается, как показано в следующей таблице, представляющей данные на конец 90-х годов.

| | Использование на практике | Теоретическая проработка | Начальный уровень спецификации |
|---|---|---|---|
| Структура | хорошо | хорошо проработана | стратегический |
| Статическая семантика | частично используется | хорошо проработана | концептуальный |
| Процессы | как-то сделано | отдельные моменты | требования |
| Динамическая семантика | отдельные части | маленькие куски | реализация |
| Сервисы | реализации | частные случаи | реализация |
| Форматы обмена | специально для конкретных систем | ничего | реализация |
| Интерфейсы | интуитивно | ничего | реализация |
| Сценарии | интуитивно | ничего | реализация |

Создание баз данных требует одновременной согласованной разработки структур, процессов, распределений и интерфейсов. Ниже мы покажем, как расширенная модель отношения элементов позволяет работать со всеми четырьмя аспектами.

В настоящее время базы данных расширяются до информационных web-систем, хранилищ данных, интеллектуальных баз знаний и систем анализа данных. Расширение можно проводить консервативным путем или на основе новых парадигм. Консервативный путь является предпочтительным, если только новые парадигмы не позволяют решать бывшие нерешенными задачи. В случае использования консервативного пути необходима хорошая архитектура [8], [6], позволяющая строить расширяемые, масштабируемые системы.

**Открытая задача 2.** *Найти архитектуру для общего расширения систем баз данных, позволяющую моделировать все службы и делать выводы о свойствах системы.*

В то же время необходимо моделировать качество работы систем баз данных. Критерии качества часто задаются весьма нечетко. Распространенными критериями качества являются точность, изменяемость, отказоустойчивость, удобство, производительность, неразглашение, восстановляемость, надежность, эффективность, безопасность, стабильность и верифицируемость [4].

**Открытая задача 3.** *Формально определить критерии качества, атрибуты и метрику для оценки качества для концептуального моделирования, а также разработать средства для внедрения, контроля и улучшения качества.*

## 2. Спецификация структур баз данных

### 2.1. Языки спецификации структур

Структура баз данных основывается на трех взаимозависимых компонентах:

**Синтаксис:** Индуктивное задание структуры с использованием множества базовых типов, набора конструкторов и теории применения конструкторов, ограничивающей применение конструкторов правилами и формулами деонтической логики. В большинстве случаев теория может быть опущена. Структурная рекурсия является основным средством задания спецификаций.

**Семантика:** Спецификация допустимых баз данных на основе статических ограничений целостности описывает легальные состояния баз данных. Если используется структурная рекурсия, для описания статических ограничений целостности может использоваться иерархическая логика предикатов первого порядка.

**Прагматика:** Описание контекста и цели основано либо на явных ссылках на модель фирмы, задачи фирмы, политику фирмы и

окружение, либо на интенциональной логике, задающей интерпретации и значения в зависимости от времени, местонахождения и здравого смысла.

Индуктивное задание структуры основано на базовых типах и конструкторах типов. Базовый тип представляет собой алгебраическую структуру $B = (Dom(B), Op(B), Pred(B))$ с именем, областью допустимых значений, множеством операций и множеством предикатов. Класс $B^C$ базового типа представляет собой набор элементов из $dom(B)$. Как правило требуется, чтобы $B^C$ было множеством. Также $B^C$ может быть списком, мультимножеством, деревом и т. п. Классы могут изменяться под воздействием операций. Элементы классов могут классифицироваться с помощью предикатов.

*Конструктор типов* представляет собой функцию из множества типов в новые типы. В состав конструктора может входить *оператор выбора* для извлечения данных (например *Select*) и *операторы обновления данных* (например *Insert, Delete, Update*) для отображения из нового типа в компоненты типов или в новый тип. Операторы обновления могут быть нагружены критериями корректности результата, правилами верификации, подразумеваемыми правилами, пользовательскими представлениями, физическими представлениями или свойствами физического представления.

В качестве примера типичных для баз данных конструкторов можно привести конструкторы *множеств, n-компонентных векторов, списков и мультимножеств*. Конструктор множеств основан на некотором определенном типе и использует алгебру операций, включающую объединение, пересечение и дополнение. Можно считать, что в этом случае оператор выбора принимает в качестве аргумента предикат. Операторы обновления, такие как *Insert* или *Delete*, определяются как выражения в алгебре множеств. В пользовательском представлении используются фигурные скобки { и }. Конструкторы типов определяют систему типов над базовыми схемами данных, то есть набор конструируемых множеств значений данных. В некоторых моделях баз данных конструкторы типов основываются на семантике указателей.

*Именование* и *ссылки* также являются удобными средствами конструирования в моделях. Каждый тип и класс концепции имеет имя. Эти имена могут быть использованы для для определения новых типов; на имена можно ссылаться при определении типа. Часто структуры включают *необязательные* компоненты. Необязательные компоненты и ссылки следует использовать с максимальной осторожностью, так как в противном случае потребуется использовать сверхсложные логики, такие как логики топов [9]. Более удачным подходом к моделированию является требование слабой идентифицируемости по значениям всех объектов баз данных [10].

## 2.2. Ограничения целостности

Ограничения целостности используются для отделения «хороших» состояний или последовательностей состояний системы баз данных от нежелательных состояний или последовательностей. Ограничения целостности используются для спецификации как семантики, так и процессов в базах данных. Следовательно непротиворечивость приложений баз данных не может рассматриваться отдельно от ограничений. В то же время ограничения целостности задаются пользователем на различных уровнях абстракции, с различными видами неопределенности и подтекстами, на разных языках. Для обработки и практического использования, однако, ограничения должны быть определены явно и однозначно. Для устранения этого противоречия пользовательские ограничения транслируются во внутрисистемные процедуры, реализующие поддержание целостности.

В каждой структуре также имеется множество **подразумеваемых наследуемых из модели ограничений целостности**.

Ограничения конструирования компонент основаны на существовании, мощности и включении компонент. Такие ограничения должны учитываться в процессе трансляции и импликации.

Ограничения идентифицируемости неявно используются в конструкторе множеств. Каждый объект должен либо не принадлежать множеству, либо входить в множество ровно один раз. Множества основаны на простых общих функциях. Свойство идентифицируемости, однако, представимо только в терминах авто-

морфизмов групп [1]. Позднее мы увидим, что представимость значений и слабая представимость значений влекут за собой контролируемость структуры.

Ацикличность и конечность структуры поддерживают аксиоматизируемость и определенность алгебры. Данные ограничения необходимо выражать явно. Ограничения типа ограничений на мощность могут основываться на потенциально бесконечных циклах.

Внешнее структурирование означает представимость ограничений посредством структуры. В этом случае сложно характеризовать импликации ограничений.

Подразумеваемые наследуемые из модели ограничения целостности относятся к фильтрам производительности и поддержания.

Ограничения целостности могут быть сформулированы в терминах BV-форм (Beeri-Vardi frames), то есть импликаций, в левой и правой и правой части которой стоят формулы. BV-ограничения, вообще говоря, не задают строгое ограничение выразимости. Если структура является иерархической, BV-ограничения могут быть заданы в рамках логики предикатов первого порядка. Можно ввести целый ряд классов ограничений целостности, таких как:

Ограничения для построения равенств позволяют для множества объектов из одного или нескольких классах генерировать равенства самих объектов или их компонент.

Ограничения для построения объектов требуют, чтобы для множества объектов, удовлетворяющих некоторому условию, существовало другое множество объектов.

Класс $\mathcal{C}$ ограничений целостности называется *замкнутым относительно импликации Гильберта*, если он может быть аксиоматизирован конечным множеством ограниченных правил вывода и конечным множеством аксиом. Известно, что множество зависимостей объединений не является замкнутым относительно импликации Гильберта для реляционных структур. Однако существует аксиоматизация с неограниченным правилом, то есть правилом с потенциально бесконечным числом посылок.

Основной проблемой является извлечение ограничений целостности. Так как необходимо обрабатывать множества, возникает необходимость в более сложных теориях вывода. Хорошим кандидатом является визуальный или графический вывод, гораздо более сильный, чем логический вывод [3].

**Открытая задача 4.** *Создать средство вывода для обработки множеств ограничений. Классифицировать «реальные» множества ограничений, которые могут быть легко заданы и поддерживаемы.*

Еще несколько проблем, связанных с зависимостями, могут быть сформулирована на уровне реляционной модели. Приведем соответствующие постановки.

**Открытая задача 5.** *Является ли проблема импликаций для зависимостей замыкания и функциональных зависимостей алгоритмически разрешимой? Является ли эта проблема аксиоматизируемой?*

*Какие подклассы ограничений по включению, содержащие унарные ограничения по включению, аксиоматизируемы вместе с классом функциональных зависимостей?*

*Какие подклассы зависимостей объединения, содержащие класс многозначных зависимостей, являются аксиоматизируемыми?*

*Охарактеризовать отношения, совместимые по функциональным зависимостям.*

*Охарактеризовать свойства классов ограничений при горизонтальной декомпозиции.*

## 2.3. Способы представлений

Классический подход к объектам баз данных заключается в хранении объектов в зависимости от их типов. Таким образом, каждая сущность оказывается представленной группой объектов, объединенных либо с помощью идентификаторов, либо с помощью специальных поддерживающих процедур. Однако, вообще говоря, следует рассматривать два различных подхода к представлению объектов:

Поклассовое представление на основе идентификаторов: Хранимые в базе данных сущности могут представляться несколькими объектами. *Идентификаторы объектов* поддерживают идентификацию без рассмотрения сложных взаимоотношений между объектами. Объекты могут быть элементами нескольких классов. На раннем этапе развития объектного подхода предполагалось, что такой класс единственный. Подобное допущение приводило к ряду проблем при миграции объектов, не имевших удовлетворительного решения.

Описанное выше представление используется в структурах на основе расширенных ER-моделей [14] и объектно-ориентированных моделей. В реляционных и объектно-реляционных системах баз данных используется следующий подход:

Пообъектное представление: В графовых моделях, разработанных для облегчения применения объектного подхода [1], объекты представлены в виде подграфов, то есть в виде совокупности вершин, ассоциированных с объектом, и соответствующих ребер. Такое представление соответствует представлению, используемому в процессе стандартизации.

XML основан на пообъектном представлении. Он позволяет использовать нулевые значения без уведомления. Если значение объекта не существует, неизвестно, неприменимо, не может быть получено и т. п., XML-схема не использует тэг, соответствующий атрибуту или компоненте. Классы являются скрытыми.

Пообъектное представление вносит существенную избыточность, которая должна поддерживаться системой, таким образом существенно снижая производительность. Помимо производительности, проблемами также являются низкая масштабируемость и неэффективное использование ресурсов. Работа с подобными системами приводит к лавинообразному появлению блокировок — любая модификация данных требует рекурсивной блокировки связанных друг с другом объектов.

Суммируя вышесказанное, пообъектное представление применимо только при выполнении следующих условий:

- Приложение изменяется мало, а структура данных и поддерживающие основные функции приложения не изменяются на протяжении жизненного цикла системы.
- Обновления данных производятся очень редко. Модификация, вставка и удаление разрешаются только внутри четко определенных ограниченных «зон» базы данных.

Типичными примерами областей применения систем с пообъектным представлением являются архивы, системы представления информации и системы управления контентом. Такие системы строятся поверх подсистемы обновления данных и называются play-out системами. Данные хранятся в том же виде, в котором они передаются пользователю. Подсистема модификации данных содержит play-out-генератор, генерирующий все необходимые просмотры данных для play-out системы.

Другим примером являются базы данных без обновлений, такие как в системе SAP, содержащей огромное число взаимозависимых просмотров.

Первое представление может быть использовано для средств поиска, второе — для ввода и вывода данных в хранилищах данных.

**Открытая задача 6.** *Построить технику и теорию для обработки избыточных множеств объектов с поддержкой управления целостностью множеств объектов, например наборов XML-документов.*

Оптимизация баз данных основывается на знании сложности операций. Если известно, что некоторое подмножество операций существенно более сложное, чем остальные операции, и известно несколько эквивалентных представлений, среди таких представлений можно найти наиболее простое. Примером оптимизации является вертикальная нормализация, ориентированная на разложение отношений на множество отношений, имеющих меньшую сложность и более простых в поддержании. Горизонтальная нормализация ориентирована

на выбор частей отношений с наименьшей сложностью. Дедуктивная нормализация ориентирована на сведение базы данных к отношениям, которые не могут быть получены из других отношений применением правил вывода. Напомним, что при нормализации получающиеся представления должны оставаться эквивалентным исходному. В настоящее время описанные виды нормализации рассматриваются по-отдельности.

**Открытая задача 7.** *Найти общую модель для применения вертикальной, горизонтальной и дедуктивной нормализации для объектно-реляционных моделей данных.*

Нормализация часто основывается на ограничениях баз данных. Для проведения корректной нормализации необходимо знать все множество ограничений на данные для рассматриваемого приложения. Но эта задача слишком сложна и часто принципиально нерешаема.

**Открытая задача 8.** *Найти теорию нормализации, применимую в случае неполноты множества ограничений.*

# 3. Спецификация функциональности

## 3.1. Операции в информационных системах

Общие операции над системами типов могут быть определены с помощью структурной рекурсии. Пусть заданы типы $T$ и $T'$, множественный тип $C^T$ над $T$ (например, множество значений типа $T$, мультимножество, список) и операции, такие как обобщенное объединение $\cup_{C^T}$, обобщенное пересечение $\cap_{C^T}$ и обобщенный пустой элемент $\varnothing_{C^T}$. Пусть также задан элемент $h_0$ типа $T'$ и две функции $h_1 : T \to T'$ и $h_2 : T' \times T' \to T'$. Тогда операция структурной рекурсии представления вставки $R^C$ над $T$ определяется следующим образом.

$srec_{h_0,h_1,h_2}(\varnothing_{C^T}) = h_0$
$srec_{h_0,h_1,h_2}(|\{|s|\}|) = h_1(s)$ для одноэлементных наборов $|\{|s|\}|$

$$srec_{h_0,h_1,h_2}(|\{|s|\}| \cup_{C^T} R^C) = h_2(h_1(s), srec_{h_0,h_1,h_2}(R^C)),$$
$$\text{если } |\{|s|\}| \cap_{C^T} R^C = \varnothing_{C^T}.$$

Все операции объектно-реляционной модели, ER-модели, и других декларативных моделей баз данных могут быть определены в терминах структурной рекурсии, например

- выбор определяется операцией $srec_{\varnothing,i_\alpha,\cup}$, где

$$i_\alpha = \begin{cases} \{o\}, & \text{если } \{o\} \models \alpha \\ \varnothing & \text{в противном случае} \end{cases}$$

- функция агрегирования может быть определена на основе двух функций для нулевых значений

$$h_f^0(s) = \begin{cases} 0, & \text{если } s = NULL \\ f(s) & \text{в противном случае} \end{cases}$$
$$h_f^{undef}(s) = \begin{cases} undef, & \text{если } s = NULL \\ f(s) & \text{в противном случае} \end{cases}$$

  и структурной рекурсии, например

$$\texttt{sum}_0^{null} = srec_{0,h_{Id}^0,+} \text{или } \texttt{sum}_{undef}^{null} = srec_{0,h_{Id}^{undef},+};$$
$$\texttt{count}_1^{null} = srec_{0,h_1^0,+} \text{или } \texttt{count}_1^{undef} = srec_{0,h_1^{undef},+}$$

  или с помощью SQL-определения, например функция вычисления среднего значения может быть представлена как

$$\frac{\texttt{sum}_0^{null}}{\texttt{count}_1^{null}}.$$

Аналогично можно определить пересечение, объединение, разность, проекцию, соединение, вложение, переименование, вставку, удаление и обновление.

Выразительная сила структурной рекурсии также ограничена. Недетерминированные программы, генерирующие вектора или объекты, не могут быть выражены в терминах структурной рекурсии.

Операции могут использоваться либо для `извлечения` значений, либо для `изменения состояния` базы данных.

Для определения операций используется подход, основанный на просмотрах, предназначенных для ограничения области значений, а также предусловий и постусловий, ограничивающих применимость, активизируемых операций, а также явного описания принудительно выполняемых операций: Операция $\varphi$

[`Просмотр:` <Имя_просмотра>]
[`Предусловие:` <Условие_активации>]
[`Активизируемая_операция:` <Спецификация>]
[`Постусловие:` <Условие_принятия>]
[`Принудительные_операции:` <Операция, Условие>]

Реляционная модель, а также объектно-реляционные модели могут быть расширены за счет добавления операций агрегирования, группировки и ограниченной рекурсии. Семантика перечисленных операций варьируется от СУБД к СУБД и до сих пор не получила математического обоснования [5].

**Открытая задача 9.** *Разработать общую теорию операций, расширяющих объектно-реляционные модели.*

### 3.2. Динамические ограничения целостности

Динамика функционирования баз данных определяется посредством систем переходов. Система переходов для схемы $S$ представляет собой пару $\mathcal{TS} = (\mathcal{S}, \{\xrightarrow{a} \mid a \in \mathcal{L}\})$ где $\mathcal{S}$ — непустое множество переменных состояния, $\mathcal{L}$ — непустое множество (меток), $\xrightarrow{a} \subseteq \mathcal{S} \times (\mathcal{S} \cup \{\infty\})$ для каждой метки $a \in \mathcal{L}$. Переменные состояния задают состояния. Переходы представляют собой транзакции в $S$.

Время жизни базы данных задается в терминах путей в $\mathcal{TS}$. *Путь* $\pi$ в системе переходов представляет собой конечную или счетную последовательность вида $s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \ldots$. Длина пути есть число переходов.

В системе переходов $\mathcal{TS}$ можно ввести *темпоральную динамическую логику баз данных*, используя кванторы $\forall_f$, (всегда в будущем), $\forall_p$ (всегда в прошлом), $\exists_f$ (в некоторый момент в будущем) и $\exists_p$ (в некоторый момент в прошлом).

Логика первого порядка может быть расширена за счет темпоральных операторов. Функция истинности расширяется за счет рассмотрения времени. Пусть имеется темпоральный класс $(R^C, l_R)$. Функция истинности $I$ расширяется за счет рассмотрения времени и определяется на $S(t_S, R^C, l_R)$. Формула $\alpha$ истинна для $I_{(R^C,l_R)}$ в момент $t_S$, если она истинна для базы данных в момент $t_S$, то есть $I_{(R^C,l_R)}(\alpha, t_S) = 1$ тогда и только тогда, когда истинна $I_{S(t_S,R^C,l_R)}(\alpha, t_S)$.

- Для формул без темпоральных префиксов расширенная истинность совпадает с обычной истинностью.
- $I(\forall_f \alpha, t_S) = 1$ тогда и только тогда, когда $I(\alpha, t_S) = 1$ для всех ${t'}_S > t_S$;
- $I(\forall_p \alpha, t_S) = 1$ тогда и только тогда, когда $I(\alpha, t_S) = 1$ для всех ${t'}_S < t_S$;
- $I(\exists_f \alpha, t_S) = 1$ тогда и только тогда, когда $I(\alpha, t_S) = 1$ для некоторого ${t'}_S > t_S$;
- $I(\exists_p \alpha, t_S) = 1$ тогда и только тогда, когда $I(\alpha, t_S) = 1$ для некоторого ${t'}_S < t_S$.

Модальные операторы $\forall_p$ и $\exists_p$ (и $\forall_f$ и $\exists_f$) являются двойственными, то есть формулы $\forall_h \alpha$ и $\not\exists_h \alpha$ эквивалентны. С помощью следующих правил описанные операторы могут быть отображены в классическую модальную логику:

$$\Box\alpha \equiv (\forall_f \alpha \wedge \forall_p \alpha \wedge \alpha);$$
$$\Diamond\alpha \equiv (\exists_f \alpha \vee \exists_p \alpha \vee \alpha).$$

Можно дополнительно ввести темпоральные операторы *until* и *next*.

Наиболее важным классом динамических ограничений целостности являются ограничения на переход $\alpha O \beta$, задающие предусловие $\alpha$ и постусловие $\beta$ для каждой операции $O$. Ограничение $\alpha O \beta$ могут быть выражено темпоральной формулой $\alpha \xrightarrow{O} \beta$.

Произвольное конечное множество статических ограничений целостности может быть эквивалентным образом записано в виде множества ограничений на переход $\{\Lambda_{\alpha \in \Sigma^\alpha} \xrightarrow{O} \Lambda_{\alpha \in \Sigma^\alpha} | O \in Alg(M)\}$.

Ограничения целостности могут накладываться:

- на процедурном уровне, с помощью:
  - введения триггеров [7] в так называемые активные настройки событие–условие–действие;
  - задания максимальных операторов, не нарушающих целостности [9];
  - хранимых процедур, то есть программ, определяющих все возможные нарушения ограничений целостности.
- на уровне транзакций, ограничивая последовательности переходов из состояния в состояние последовательностями, не нарушающими ограничения целостности;
- на уровне СУБД, на основе декларативных спецификаций, зависящих от возможностей СУБД;
- на уровне интерфейса, путем рассмотрения операций, не нарушающих целостность.

Ограничения на базы данных отображаются в ограничения на переходы. Ограничения на переходы хорошо изучены, прежде всего благодаря их локальности. Такие ограничения могут поддерживаться с помощью триггеров или хранимых процедур. Однако вопрос глобальных взаимозависимостей остается открытым.

**Открытая задача 10.** *Разработать теорию взаимного влияния ограничений целостности баз данных, отображаемого на удобные множества триггеров и хранимых процедур.*

### 3.3. Задание последовательности выполняемых действий

В литературе предлагалось значительное количество подходов к заданию последовательности выполняемых действий. С нашей точки зрения предпочтительным является формальное описание с графическим представлением, позволяющее избегать проблем, связанных с методами чисто графического задания, такими как и/или ловушки. Рассмотрим алгебру базовых шагов вычисления, введенную в [16].

- Базовыми управляющими командами являются последовательное выполнение ; (выполнение шагов по очереди), распараллеливание $|\wedge|$ (выполнение шагов в параллельном режиме), исключающий выбор $|\oplus|$ (выбор одного пути выполнения из нескольких возможных), синхронизация $|sync|$ (синхронизация двух параллельных путей выполнения с помощью синхронизирующего условия $sync$), и простое слияние + (слияние двух параллельных путей выполнения). Исключающий выбор является подразумеваемой параллельной операцией и обозначается $||$.
- Структурными управляющими командами являются произвольные циклы $*$ (выполнение шагов без каких-либо структурных ограничений на циклы), произвольные циклы + (выполнение шагов без каких-либо структурных ограничений на циклы, за исключением того, что цикл должн быть пройден по крайней мере один раз), опциональное выполнение [] (выполнение шага один раз или пропуск шага), неявное прерывание $\downarrow$ (закончить выполнение, если больше не осталось шагов), вход в подшаг $\nearrow$ и завершение подшага $\searrow$.

Расширим алгебру с помощью дополнительного набора команд.

- Дополнительные команды ветвления и синхронизации включают в себя множественный выбор $|(m,n)|$ (выбрать из всего множества возможных путей выполнения от $m$ до $n$ путей), множественное слияние (слияние нескольких путей выполнения без синхронизации), дискриминатор (слияние нескольких путей выполнения без синхронизации с выполнением последующих шагов не более одного раза), объединение $n$ из $m$ (слияние нескольких путей выполнения с частичной синхронизацией и выполнением следующего шага не более одного раза), и синхронизирующее объединение (слияние нескольких путей выполнения; если путей несколько, производится синхронизация, в противном случае выполняется простое слияние).
- Можно также рассмотреть управляющие команды на множестве объектов (CMO-команды), такие как CMO-команды с получением информации на этапе проектирования (сгенерировать множество реализаций одного шага, причем мощность множества

определяется на этапе проектирования), CMO-команды с получением информации на этапе выполнения (сгенерировать множество реализаций одного шага, причем мощность множества определяется в некоторый момент выполнения, как, например, в FOR-циклах), CMO-команды без получения предварительной информации (сгенерировать множество реализаций одного шага, причем мощность множества неизвестна, как, например, в WHILE-циклах), и CMO-команды с синхронизацией (синхронизирующие ребра, создать множество реализаций одного действия и провести синхронизацию после его выполнения).

- Управляющие команды на основе состояний включают отклоняющий выбор (выполнить один из двух путей, причем выбирается подразумеваемый путь), наложенное параллельное выполнение (выполнить два действия в случайном порядке, но не в параллель, а последовательно), и предел (выполнение действий вплоть до достижения некоторого предела).

- Отменяющие команды включают в себя отмену шага, отмену ветви и т. п.

Описанные выше операторы являются обобщениями шаблонов последовательностей выполняемых действий, созданными в соответствии с подходами, разработанными для алгебр сетей Петри.

Операции, определенные с помощью описанной идеологии, могут быть непосредственно оттранслированы в программы для баз данных. На данный момент не существует теории поведения баз данных, которая могла бы охватить поведение баз данных во всей широте и глубине. Отправной точкой для создание такой теории, возможно, станет изложенное в работе [15] предложение использовать абстрактные машины состояний [2].

**Открытая задача 11.** *Разработать теорию поведения баз данных. Теория должна охватывать работу как самой базы данных, так и системы управления базой данных.*

### 3.4. Архитектура СУБД

Функционирование информационных систем моделируется с помощью декомпозиции множества состояний системы на четыре типа состояний:

$$\mathcal{E}R^C = \begin{array}{l}(\text{входные состояния } \mathcal{IN}, \text{выходные состояния } \mathcal{OUT},\\ \text{состояния СУБД } \mathcal{DBMS}, \text{состояния базы данных } \mathcal{DB})\end{array}$$

Входные состояния охватывают входную информацию системы, то есть запросы и данные. Выходные состояния отражают вывод СУБД, то есть выходную информацию и сообщения об ошибках. Состояния СУБД содержат внутренние состояния системы управления. Состояния базы данных описывают содержимое базы. Все четыре класса состояний могут быть структурированы. Например, если состояние базы данных структурируются в соответствии со схемой данных, входные состояния структурируются аналогично.

При использовании ориентированных на значения или объектно-реляционных моделей состояния баз данных могут быть представлены отношениями. В этом случае обновление одного из типов схемы отображается в изменение одного из отношений. Изменения состояний моделируются с помощью *правил изменения состояний* абстрактных машин состояний [2]. СУБД задается программой и управлением. Под программой мы будем понимать элементы работы или сервисов, удовлетворяющие заданным критериям качества, под управлением и координацией — манипулирование блоками программ, возможно с дополнительными требованиями *атомарности и непротиворечивости*. Также управление и координация могут задаваться с помощью *команд управления задачами*.

При вызове программ производится инициализация значений входных переменных. Переменные могут быть статическими, храниться на стеке, задаваться явно или неявно. Дополнительно могут использоваться такие параметры вызовов, как `onSubmit` (ввод значения) и `presentationMode` (режим презентации), параметры приоритета `onFocus` (фокусировка на процессе) и `emphasisMode` (режим выделения), параметры управления `onRecovery` (восстановление после сбоя) и `hookOnProcess` (активация процесса), параметры ошибок

`onError` (рассмотрение ошибок) и `notifyMode` (режим уведомлений), а также общие параметры передачи, такие как `onReceive` (активация приема) и `validUntil` (ограничение времени жизни значения).

Требования атомарности и непротиворечивости поддерживаются в рамках ряда моделей транзакций. В качестве примера можно привести плоские транзакции, сага-транзакции, контракты и т. п. [14].

Пусть $T'$ — подтип СУБД $\mathcal{E}R^C$. Рассмотрим изменение $T(s_1, \ldots, s_n) := t$. Множество $\mathcal{U} = \{T_i(s_{i,1}, \ldots, s_{i,n_i}) := o_i | 1 \leqslant i \leqslant m\}$ объектно-ориентированных изменений состояний называется непротиворечивым, если для всех $1 \leqslant i < j \leqslant m$ из равенства $T_i(s_{i,1}, \ldots, s_{i,n_i}) = T_j(s_{j,1}, \ldots, s_{j,n_j})$ следует равенство $o_i = o_j$.

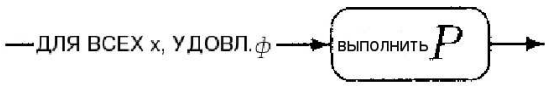
*Результатом* выполнения непротиворечивого множества $\mathcal{U}$ изменений состояний из $\mathcal{E}R^C$ является новое состояние $\mathcal{E}R^C + \mathcal{U}$, для каждого объекта $o$ из $\mathcal{E}R^C$ определяемое по формуле

$$(\mathcal{E}R^C + \mathcal{U})(o) = \begin{cases} \mathit{Update}(T_i, s_{i,1}, \ldots, s_{i,n_i}, o_i), \\ \quad \text{если } T_i(s_{i,1}, \ldots, s_{i,n_i}) := o_i \in \mathcal{U} \\ \mathcal{E}R^C(o) \\ \quad \text{в противном случае.} \end{cases}$$

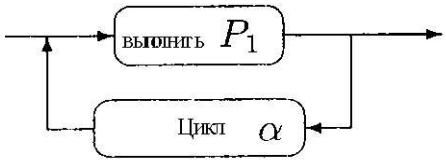
Параметризованная программа $r(x_1, \ldots, x_n) = P$ арности $n$ состоит из имени $r$, правила перехода $P$ и множества свободных переменных $\{x_1, \ldots, x_n\}$ правила $P$.

Информационная система $\mathcal{E}R^C$ является моделью формулы $\phi$ ($\mathcal{E}R^C \models \phi$), если $[[\phi]]_\zeta^{\mathcal{E}R^C} =$ истина для всех возможных значений $\zeta$ свободных переменных $\phi$.

Рассмотрим две типичных конструкции для порождения программ. Выполнение программы для всех значений, удовлетворяющих некоторому условию, может быть представлена следующей схемой:




Выполнение программы как шага цикла может быть представлено следующей схемой:




Дополнительно рассмотрим такие конструкторы, как последовательное выполнение, ветвление, параллельное выполнение, выполнение после присваивания значений, выполнение после выбора произвольного значения, переход на следующий шаг, изменение состояния информационной системы и вызов подпрограммы.

Используем абстрактные машины состояний и для определения семантики программ.

Правило перехода $\mathcal{P}$ порождает множество изменений состояний $\mathcal{U}$ в состоянии $\mathcal{E}R^C$, если $\mathcal{U}$ непротиворечиво. Происходит изменение состояния информационной системы с присваиванием значений переменных $\zeta$, обозначаемое $\texttt{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta, \mathcal{U})$.

Семантика правил перехода определяется с помощью исчисления правил следующего вида:

$$\frac{\text{предусловие}_1, \ldots, \text{предусловие}_n}{\text{вывод}} \text{ где условие.}$$

Например, изменение состояния, заданное первым из описанных конструкторов, определяется правилом

$$\frac{\forall a \in I : \texttt{yields}(\mathcal{P}, \mathcal{E}R^C, \zeta[x \mapsto a], \mathcal{U}_a)}{\texttt{yields}(\text{ДЛЯ ВСЕХ } x,\ \text{УДОВЛ. } \phi,\ \text{ВЫПОЛН. } \mathcal{P}, \mathcal{E}R^C, \zeta, \cup_{a \in I}\mathcal{U}_a)}$$
$$\text{где } I = \mathit{range}(x, \phi, \mathcal{E}R^C, \zeta).$$

Диапазон $\mathit{range}(x, \phi, \mathcal{E}R^C, \zeta)$ представляет собой множество $\{o \in \mathcal{E}R^C | [[\phi]]^{\mathcal{E}R^C}_{\zeta[x \to a]} = \text{истина}\}$.

**Открытая задача 12.** *Разработать общую теорию абстракций и уточнений для систем баз данных, поддерживающую различные архитектуры и позволяющую декомпозировать базы данных на компоненты.*

## 4. Задание распределения

Задача задания распределения долгое время игнорировалась. Явное задание распределения не применялось, вместо этого использовались различные подходы моделирования сотрудничающих систем, такие как системы нескольких баз данных или союзничающие системы баз данных.

### 4.1. Комплект просмотров

По классическому определению, (простой) просмотр представляет собой одноэлементный тип, данные которого выбираются из базы по некоторому запросу вида

```
создать просмотр <имя> (<переменные, на которые проецируется
                                                    просмотр>)
выбрать           <выражение, задающее проекцию>
из                <подсхема базы данных>
где               <условие выбора>
сгруппировать по  <выражение для группировки>
с                 <выбор из групп>
упорядочить по    <порядок выдачи информации в просмотре>
```

Так как возможно использование поклассового представления, простые просмотры могут оказаться неоптимальной структурой для спецификации обмена информацией. Предпочтительной моделью является комплект просмотров. *Комплект* состоит из множества элементов, схемы интеграции или ассоциирования элементов и требований к поддержанию отношения ассоциирования.

Простые примеры комплектов просмотров рассмотрены в работе [14], где в качестве комплектов выступают ER-схемы. Интеграция задается схемой. Требования к поддержанию основаны на концепции «главный-подчиненный», то есть состояние классов комплекта просмотров изменяется при изменении соответствующих областей базы данных.

Просмотры должны также реализовывать поддержку сервисов. Сервисы предоставляют свои данные и функциональность. Подобная объектная ориентированность удобна, если возникает необходимость

использования данных без установления прямого или удаленного соединения с СУБД.

Рассмотрим обобщенную форму задания просмотра для реляционных баз данных:

сгенерировать <отображение: переменные → выходная структура>
из <типы базы данных>
где <условие выбора>
представление с использованием <общий стиль представления>
& <абстракция (гранулярность, мера, точность)>
& <упорядочение в рамках представления>
& <иерархические представления>
& <точки просмотра>
& <разделения>
с учетом определений <условие>
& <перемещение>
с использованием функций <функции поиска>
& <функции экспорта данных>
& <функции ввода данных>
& <функции сессии>
& <функции разметки>

Расширение просмотров за счет функций кажется избыточным на этапе проектирования баз данных. Использование просмотров в распределенных средах удается минимизировать усилия, связанные с параллельной и последовательной разработкой, так как сразу генерируется комплект просмотров вместо того, чтобы отдельно разрабатывать каждый просмотр.

## 4.2. Сервисы

Традиционно сервисы изучались в рамках одного из (семи) уровней коммуникационных систем и характеризовались двумя параметрами: функциональностью и качеством обслуживания. Однако мы используем более современный подход [8] и будем рассматривать не функции, а *информационные процессы. Качество обслуживания* ограничивается рядом свойств, формулируемых или на уровне реализации, или на уровне концепции, или на уровне пользовате-

лей. Сервис состоит из информационного процесса, предоставляемых характеристик и свойств, гарантирующих качество обслуживания. Формально сервис представляет собой тройку $\mathcal{S} = (\mathcal{I}, \mathcal{F}, \Sigma_{\mathcal{S}})$, где $\mathcal{I} = (\mathcal{V}, \mathcal{M}, \Sigma_{\mathcal{I}})$.

Информационный процесс задается тремя компонентами:

Просмотры из комплекта $\mathcal{V}$ являются ресурсами информационного процесса. Так как просмотры расширены за счет функций, они обладают вычислительными возможностями и могут использоваться в качестве статистических пакетов, хранилищ данных или средств раскопок данных.

Менеджер сервиса $\mathcal{M}$ поддерживает функциональность и качество обслуживания и управляет контейнерами, их play-out-функциями и доставкой информации клиентам. Менеджеры сервисов также называются провайдерами сервисов.

Область применимости сервиса задается в виде множества допустимых задач $\mathcal{T}$.

Характеристики сервиса $\mathcal{F}$ задаются в зависимости от уровня абстракции.

Характеристики на уровне пользователей основаны на соглашениях уровня сервиса и информационных процессах этого уровня.

Характеристики на уровне концепции описывают свойства, которыми сервис должен обладать для выполнения соглашений уровня сервиса. Здесь же задаются доступные клиентам функции путем спецификации интерфейсов и семантики.

Характеристики на уровне реализации описывают синтаксические интерфейсы функций, предоставляемые данные, поведение и ограничения на информационную систему и клиентов.

Качество обслуживания $\Sigma_{\mathcal{S}}$ задается в зависимости от уровня абстракции.

Параметры качества на уровне пользователей включают всеохватность (неограниченность доступа в пространстве и времени), и безопасность (по отношению к сбоям, атакам, ошибкам; степень доверенности).

Параметры качества на уровне концепции включают интероперабельность (формальный каркас для интерпретации) и непротиворечивость (функций данных).

Параметры качества на уровне реализации включают долговечность (доступ ко всей информации, если не оговорено противное), надежность (на основе моделей сбоев для устойчивости, конфликтов и живучести), производительность (на основе модели расходов, времени отклика и пропускной способности), и масштабируемость (по отношению к изменениям сервисов, числа клиентов и серверов).

### 4.3. Формы обмена информацией

Форма обмена информацией определяется следующими параметрами.

Архитектура обмена обычно основывается на архитектуре системы, интегрирующей информационные системы с помощью подсистем коммуникаций и обмена.

Стиль сотрудничества задает поддерживающие программы, стиль взаимодействия и координирующие средства.

Шаблон сотрудничества задает роли партнеров, права и обязанности и протоколы общения.

Распределенные системы баз данных основываются на локальных системах баз данных, объединенных с помощью заданной стратегией интеграции. Основой интеграции является полное объединение локальных концептуальных схем в глобальную схему распределения. Архитектура модели представлена на рис. 1.

Помимо классических распределенных систем можно также рассмотреть и другие архитектуры, такие как *фермы баз данных, наращиваемые сообщества информационных систем* и *сотрудничающие информационные системы*. Последняя модель основана на концепции сотрудничающих просмотров [14]. Наращиваемые сообщества информационных систем являются основой систем управления оборудованием. Простыми примерами таких систем являются хранилища данных и средства управления контентом.


Рис. 1. Обобщение трехуровневой архитектуры на случай распределенной схемы.

Фермы баз данных являются обобщением и расширением подхода, заложенного в союзничающие информационные системы и системы-посредники. Архитектура ферм баз данных изображена на рис. 2. Фермы основаны на подходе к соразработке и концепциях информационного элемента и контейнера.

Информационные элементы — это обобщенные просмотры. Просмотры генерируются на основе содержимого базы данных. Элементы — это просмотры, функциональность которых расширена, чтобы использовать собственные данные. Информационные элементы могут быть ориентированы либо на *извлечение*, либо


Рис. 2. Ферма баз данных.

на *модификацию данных*. Элементы первого типа используются для введения новых данных, элементы второго типа — для модификации локальных бах данных.

Контейнеры поддерживают экспорт и импорт данных с помощью информационных элементов, предоставленных состояниями просмотров. Элементы объединяются в контейнеры, которые могут быть подгружены или выгружены специальным образом. Процедура выгрузки поддерживает диалоговые сцены и шаги.

Система глобальных коммуникаций и ферм предоставляет протоколы обмена, средства загрузки и выгрузки контейнеров и средства модификации элементов данных, ориентированных на модификацию.

Задача полного объединения локальных баз данных не ставится. Задача заключается в создании *сотрудничающих просмотров*.

Архитектура обмена может включать в себя рабочие области клиентов, описывающие *участников, группы, роли* и права участников в рамках групп, *пакет задач* и *организацию* сотрудничества, коммуникации и взаимодействие.

Стиль сотрудничества определяется четырьмя компонентами:

поддерживающими программами информационных систем, включающими управление сессиями, управление пользователями и систему тарификации и оплаты;

шаблонами доступа для *передачи* данных через сеть, например многоадресная или одноадресная передача, для *разделения* ресурсов, на основе либо моделей транзакций, согласия и восстановления, либо репликации с управлением сбоями, и для *удаленного доступа*, включая планировщик доступа;

стилем взаимодействия на основе одноранговых моделей или компонентных моделей или моделей инициирования событий с ограничением на возможные коммуникации;

координацией последовательностей выполняемых действий, описывающей сотрудничество партнеров, типы диалогов, отображение пространств имен и правила сотрудничества.

Известен целый ряд шаблонов сотрудничества, поддерживающих *доступ* и *конфигурирование* (оборачивающая фронтальная компонента, конфигурирование компонент, перехватчик, расширяющие интерфейсы), *обработку событий* (пререагирование, постреагирование, асинхронные токены завершения, соединение приема), *синхронизацию* (блокирование подобластей, блокирование с заданной стратегией, интерфейсы безопасного взаимодействия многонитевых программ, оптимизация блокировок с двойной проверкой) и *параллельное выполнение* (активные объекты, мониторинг, полусинхронное полуасинхронное выполнение, ведущий/ведомый, определяемое нитями хранение).

Сотрудничество на основе проксирования использует частичные копии системы (удаленный прокси-сервер, прокси-сервер защиты, кэширующий прокси-сервер, синхронизирующий прокси-сервер и т. д.)

Сотрудничество на основе посредников поддерживает координацию коммуникаций, осуществляемую напрямую, через передачу сообщений, на основе концепций торговли, с помощью систем адаптеров-посредников, или с помощью систем отзыва посредников.

Сотрудничество на основе отношения главный/подчиненный использует жесткое реплицирование данных в соответствии с различными прикладными сценариями (отказоустойчивость, параллельное выполнение, улучшение точности; реплицирование процессов или нитей; реплицирование с координацией или без координации).

Сотрудничество на основе отношения клиент/координатор основано на пространствах имен и их отображении.

Сотрудничество на основе отношения издатель/подписчик также иногда называется концепцией зависимости от наблюдателя. Могут рассматриваться как активные, так и пассивные подписчики. У каждого подписчика имеется профиль подписки.

Сотрудничество на основе отношения модель/просмотр/управление аналогично трехуровневой архитектуре систем баз данных. Просмотры и управление задают интерфейсы.

Шаблоны сотрудничества обобщают протоколы за счет включение в рассмотрение партнеров по процессу, их права, ответственности и роли.

## 5. Спецификация интерактивности

Интерактивность информационных систем как правило рассматривается на уровне систем представления, путем архитектурного или Seeheim-разделения системы приложений и системы представления. Структура и функциональность задаются в терминах языка моделирования баз данных и соответствующей алгебры. Прагматика как правило не рассматривается в рамках моделей баз данных. Интерактивность по отношению к системе приложений основывается на множестве просмотров, определенных на структуре базы данных и поддерживаемых некоторой функциональной базой.

Общая архитектура информационной web-системы представлена на рис. 3. Данная архитектура успешно применялась в более чем 30 проектах по созданию огромных или очень больший web-сайтов с интенсивной информационной загрузкой и в более чем 100 проектах по созданию больших информационных систем.


Рис. 3. Подход к спецификации информационных систем.

В рамках подхода к соразработке данный подход обобщен за счет следующих добавлений:

введение новых типов объектов (медиа-объектов), по сути являющихся обобщенными просмотрами, расширенными с помощью соответствующей функциональности, адаптированными к нуждам пользователей и доставляемых участникам с помощью контейнеров [11];

введение пространств сценариев [12], задающих сценарии использования ресурсов различными группами пользователей (называемых участниками) в рамках некоторого контекста. Сценарии могут генерироваться на основе реальных действий и с помощью различных play-out-средств.

Модель взаимодействия с пользователями включает нескольких партнеров (сгруппированных по некоторым признакам; представители групп называются участниками), рассматривает разнообразные действия и описывает взаимозависимости действий. Помимо последовательности шагов взаимодействия, контента шагов взаимодействия и формы взаимодействия модель охватывает также окружение системы, задачи и участников.

### 5.1. Пространство сценариев

Модели взаимодействия должны поддерживать множество сценариев. При этом необходимо учитывать профили и окружение пользователей. *Сценарий* взаимодействия представляет собой сюжет рассказа о работе пользователя или перечень событий. Язык SiteLang [16] предоставляет средства для определения пространств сценариев, сцен и сюжетных линий в рамках сценариев.

В рамках сценария можно выделить нити активности, называемые *сюжетными линиями*, то есть пути, составленные из сцен и переходов между сценами. Пространство сценариев есть семерка $\Sigma_W = (S_W, T_W, E_W, G_W, A_W, \lambda_W, \kappa_W)$, где $S_W, T_W, E_W, G_W$ и $A_W$ — множество сцен, созданных $W$, множество переходов, множество возможных событий, множество средств защиты и множество действий $W$, соответственно. Таким образом, $T_W$ является подмножеством $S_W \times S_W$. Далее, $\lambda_W : S_W \to \mathit{SceneSpec}$ — функция, ассоциирующая с каждой сценой ее спецификацию, а $\kappa_W : T_W \to E_W \times G_W \times A_W$, $t \mapsto (e, g, a)$ — функция, ассоциирующая с каждым переходом $t$ событие $e$, инициирующего переход $t$, средство защиты $g$ (то есть логическое условие, блокирующее переход в случае ложности при событии $e$) и действие $a$, выполняемое при переходе.

Сцены представляют собой точки, в которых происходит взаимодействие, то есть диалог. Диалоги могут задаваться с помощью так называемых выражений шагов диалога. Каждая сцена обладает уникальным идентификатором Идентификатор_Сцены. С каждой сценой ассоциируется медиа-объект, множество вовлеченных участников, спецификация представления и контекст. Таким образом, сцена представляет собой следующую структуру:

Сцена = (Идентификатор_Сцены
    Выражение шагов диалога
    Пользователь
        Идентификатор_Пользователя
        Права_Пользователя
        Задачи_Пользователя
        Роли_Пользователя
    Представление (стили, подразумеваемые значения, ...)
    Контекст (оборудование, канал, подробности).

Выражение шагов диалога состоит из диалогов и примененных к диалогам операторов. Типичная сцена представлена на рис. 4. Участник соревнования по поиску данных может ввести свои решения. Для участия в соревновании необходимо внести организационный взнос. Система может знать, а может и не знать пользователя и его профиль. Если пользователь уже внес организационный взнос, диалог оплаты не выводится. Если же пользователь не внес взнос или неизвестен системе, диалог ввода решений доступен только после успешного окончания диалога оплаты.

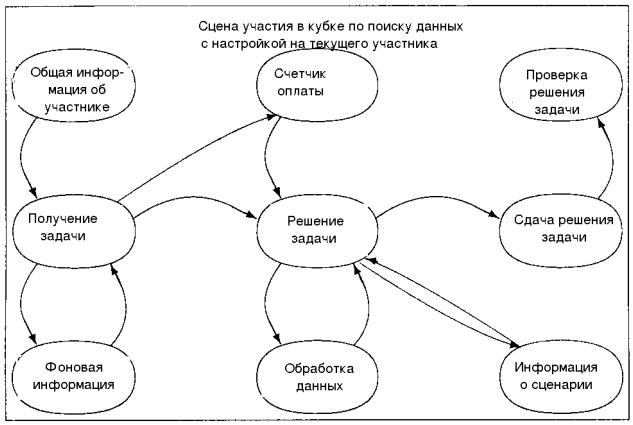



Рис. 4. Одна из сцен для активного обучения.

### 5.2. Комплект медиа-типов

Медиа-типы были введены в работе [11]. Так как разным пользователям в зависимости от истории работы, профиля и окружения требуются существенно отличающиеся данные, пакеты данных передаются через контейнеры. Контейнеры обладают всей функциональностью комплектов просмотров. Комплекты медиа-типов основаны на комплектах просмотров, снабженных специальными средствами доставки и извлечения. Комплекты медиа-типов управляются системой, состоящей из трех компонент:

Система извлечения медиа-объектов: Медиа-объекты извлекаются и удаляются из базы данных или базы знаний, обобщаются и встраиваются в другие медиа-объекты.

Система хранения и поиска медиа-объектов: Медиа-объекты могут генерироваться налету, когда требуется обратиться к их содержимому, или храниться в подсистеме хранения и поиска. Так как порождение медиа-объектов как правило имеет высокую сложность, и для нормального функционирование требуется поддержание нескольких версий, предпочтительным способом является хранение.

Система доставки медиа-объектов: Медиа-объекты используются в разнообразных задачах, разнообразными пользователями в различных социальных и организационных контекстах, в разнообразных окружениях. Система доставки медиа-объектов осуществляет доставку медиа-объектов пользователям в требуемой форме. Контейнеры содержат и управляют множеством медиа-объектов, доставляемых одному пользователю. Пользователь получает адаптированный под него контейнер и может использовать этот контейнер в своей локальной базе данных.

Описанный подход очень близок к концепции хранилищ данных. Он также основан на классической концепции модель–просмотр–управление. Концепция обобщена на медиа-объекты, которые могут просматриваться различными способами, могут генерироваться и управляться генераторами.

# 6. Интегрирование спецификационных аспектов в соразработку

Описанные выше языки довольно сложны, а непротиворечивая разработка всех аспектов информационных систем трудна. Мы разработали ряд методологий, позволяющих обойти ряд трудностей, связанных с непротиворечивой и полной разработкой. Основой методологий является проектирование сверху вниз (поэтапное уточнение), разделяющее интересующие аспекты на несколько уровней абстракции и использующее операции расширения, детализации, реструктурирования и уточнения.

## 6.1. Модель уровней абстракции для разработки информационных систем

Информационная система может быть задана на разных уровнях абстракции (рис. 5).

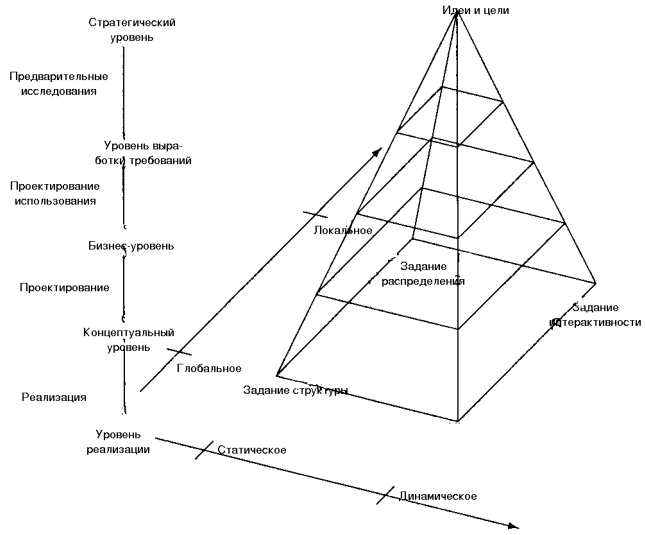



Рис. 5. Модель уровней абстракции процесса разработки баз данных.

1) *Стратегический уровень* моделирует цели информационной системы, то есть основную задачу и предполагаемые типы клиентов и решаемые ими задачи. Результатом проектирования на стратегическом уровне является *спецификация организационного контракта*.
2) *Уровень выработки требований* описывает информационную систему, анализирует бизнес-процессы и цели для формулирования требований к информационной системе. Результатом проектирования на уровне выработки требований является *спецификация системы*.
3) *Бизнес-уровень* моделирует предполагаемое использование информационной системы в терминах типов клиентов, месторасположения пространств информации, переходов между пространствами, а также диалогов между пользователями различных категорий (называемых участниками). Результатом проектирования на уровне бизнес-процессов является *расширенное руководство по системе*, включающее макеты интерфейсов и сценарии использования.
4) *Концептуальный уровень* ориентирован на интегрирование концептуальных спецификаций структуры, функциональности, распределения и интерактивности. Результатами проектирования на концептуальном уровне являются схема базы данных, последовательности выполняемых действий, комплекты просмотров и медиа-типов, спецификация сервисов и форматов обмена, а также сценарии.
5) *Уровень реализации* ориентирован на спецификацию логических и физических структур баз данных, процедур поддержания целостности, программ и интерфейсов. Спецификация осуществляется в рамках языков выбранной платформы. Результатом проектирования на уровне реализации является *модель реализации*. Модель существенно зависит от создателя информационной системы.
6) *Уровень использования* здесь не рассматривается. Поддержка, обучение, введение и администрирование как правило находятся вне сферы концептуального моделирования приложений.

## 6.2. Методология соразработки

Методологии должны соответствовать требованиям к непротиворечивой разработке систем SPICE версии 2.0 и SW-CMM версии 2.0. Соразработка основана на пошаговом уточнении системы при движении по уровням абстракции. Так как четыре аспекта информационных систем — структура, функциональность, распределение и интерактивность — взаимозависимы, они не могут разрабатываться отдельно друг от друга. Приведенная ниже методология основана на шагах следующей структуры:

Используются следующие шаги:
*Стратегический* уровень

1) Разработка видения, целей и задач
2) Анализ проблем и конкурентов

Уровень *выработки требований*

3) Разбиение на компоненты
4) Создание наброска пространства сценариев
5) Создание наброска комплекта просмотров
6) Задание бизнес-процессов

*Бизнес*-уровень

7) Разработка сюжетных линий в пространстве сценариев
8) Выявление основных типов данных и ассоциаций между ними
9) Разработка ядра ограничений целостности, например ограничений идентификации
10) Задание действий пользователей, требований используемости, и создание наброска медиа-типов
11) Выявление требований всеохватности и безопасности

*Концептуальный* уровень

12) Задание пространства сценариев
13) Разработка типов данных, ограничений целостности, их реализации

| **Правило #i**<br>**Имя шага** | Задача 1.<br>Задача 2.<br>... |
|---|---|
| Используемые документы | Документы предыдущих шагов<br>(документы по разработке системы)<br>Документация и информация от клиентов |
| Изменяемые документы | Документы по разработке системы<br>Контракты |
| Задачи и цели | Общие задачи шага<br>Согласованные цели шага<br>Основная цель |
| Вовлеченные участники | Участник A, например представители клиента<br>Участник B, например разработчик |
| Теоретические основы | Теория баз данных<br>Теория организации<br>Информатика<br>Теория познания, психология, педагогика |
| Методы и эвристики | Синтаксис и прагматика<br>Используемые языки спецификаций<br>Подходы к упрощению |
| Разработанные документы | Документы по разработке системы |
| Результаты | Результаты, в том числе поставляемые клиенту |
| Условия начала шага | Условия наличия информации,<br>выполняемые на стороне клиентов<br>Условия зависимости информации<br>Условия на участие |
| Условия завершения шага | Полнота и правильность критериев<br>Подписанные бумаги, контракты<br>Критерии качества<br>Выполнение задач шага |

14) Задание комплекта просмотров, сервисов и форматов обмена
15) Проектирование последовательностей выполняемых действий
16) Контроль результатов на тестовых данных, тестовых процессах и тестовых сюжетных линиях

17) Задание комплекта медиа-типов
18) Модулярное уточнение типов, просмотров, операций, сервисов и сцен
19) Нормализация структуры
20) Интеграция компонент по всей архитектуре

*Требования реализации*

21) Преобразование концептуальных схем в логические схемы, программы и интерфейсы
22) Разработка логических сервисов и форматов обмена
23) Разработка решений для повышение производительности, настройка
24) Преобразование логических схем в физические схемы
25) Проверка долговечности, надежности, масштабируемости и расширяемости

Методология соразработки использовалась на практике для реализации большого количества информационных систем и вместе с тем имеет надежную теоретическую основу. Нашей задачей является не конкуренция с UML, а поддержка разработки систем на твердой основе, без неясностей, пропусков и несочетаемых концепций.

## 7. Заключение

Исследования баз данных и информационных систем привели к созданию технологии, ставшей частью современной инфраструктуры. Базы данных используются как встроенные системы, например в автомобильных навигационных программах, как совокупность сотрудничающих систем и как одиночные системы. Технология достаточно хорошо отработана для того, чтобы устанавливать системы баз данных в любых приложениях с поддержкой вычислений, основанных на обработке большого объема информации. Современные информационные системы часто реализуют поддержку распределенных вычислений и web-технологии. Новые архитектуры подняли новые

проблемы, которые в настоящее время являются предметом интенсивных исследований. В работе показано, как классическая теория баз данных может быть расширена для решения описанных задач. Отметим, что предлагаемый подход является одним из возможных, могут применяться и другие решения.

В то же время в области исследования баз данных остаются и открытые задачи. В работе приведен обзор современных достижений в области теории баз данных, в основном связанных с вопросами структуры и функциональности. Некоторые результаты уже неприменимы к новым моделям информационных систем. Однако остается возможность встраивания классических компонент в новые модели. В первой части работы приведен обзор основных результатов и постановки открытых задач.

## Список литературы

[1] Beeri C., Thalheim B. Identification as a primitive of datavase models // Proc. Fundamentals of Information Systems, 7th Int. Workshop on Foundations of Models and Languages for Data and Objects — FoMLaDO'98 (Timmel, Ost-friesland, 1999) / T. Polle, T. Ripke, and K.-D. Schewe, eds. London: Kluwer. P. 19–36.

[2] Börger E., Stärk R. Abstract state machines — A method for high-level design and analysis. Berlin: Springer, 2003.

[3] Demetrovics J., Molnar A., Thalheim B. Graphical and spread-sheet reasoning for sets of functional dependencies // In ER'2004 (2004). LNCS 3255. P. 54–66.

[4] Jaakkola H. Software quality and life cycles // In ADBIS'05 (Tallinn, September 2005). Springer.

[5] Lenz H.-J., Thalheim B. Olap databases and aggregation functions // In 13th SSDBM 2001 (2001). P. 91–100.

[6] Lenz H.-J., Thalheim B. OLTP-OLAP schemes for sound applications // In TEAA 2005 (Trondheim, 2005). Vol. LNCS 3888. Springer. P. 99–113.

[7] Levene M., Loizou G. A guided tour of relational databases and beyond. Berlin: Springer, 1999.

[8] Lockermann P. Information system architectures: From art to science // Proc. BTW'2003. Berlin: Springer, 2003. P. 1–27.

[9] Schewe K.-D. The specification of data-intensive application systems / PhD thesis. Brandenburg University of Technology at Cottbus, Faculty of Mathematics, Natural Sciences and Computer Science, 1994. Advanced PhD Thesis.

[10] Schewe K.-D., Thalheim B. Fundamental concepts of object oriented databases // Acta Cybernetica. 11. 4. 1993. P. 49–81.

[11] Schewe K.-D., Thalheim B. Modeling interaction and media objects // NLDB. Natural Language Processing and Information Systems, 5th Int. Conf. on Applications of Natural Language to Information Systems. NLDB 2000, Versailles, France, Jun 28–30, 2000. Revised Papers (2001) / M. Bouzeghoub, Z. Kedad, and E. Métais, eds. Vol. 1959 of LNCS. Springer. P. 313–324.

[12] Srinivasa S. A calculus of fixpoints for characterizing interactive behavior of information systems / PhD thesis. Brandenburg University of Technology at Cottbus, Faculty of Mathematics, Natural Sciences and Computer Science, 2001.

[13] Thalheim B. Open problems in relational database theory // Bull. EATCS 32 (1987). P. 336–337.

[14] Thalheim B. Entity-relationship modeling — Foundations of database technology. Berlin: Springer, 2000. См. также http://www.is.informatik.uni-kiel.de/ thalheim/HERM.htm.

[15] Thalheim B. ASM specification of internet information services // Proc. Eurocast 2001, Las Palmas (2001). P. 301–304.

[16] Thalheim B., Düsterhöft A. Sitelang: Conceptual modeling of internet sites // ER (2001). H. S. Kunii, S. Jajodia, A. Solvberg, eds. Vol. 2224 of LNCS. Springer. P. 179–192.

**Замечание.** Основной задачей работы было проведение обзора текущего состояния исследований в области баз данных. В библиографию включены только источники, на которые есть ссылки в работе. Обширная библиография по тематике работы содержится, например, в [14].