TRANSPORTA UN SAKARU INSTITŪTS
DATORZINĀTŅU DEPARTAMENTS
magistrants Aleksejs Guļajevs
4601MV
magistrants Aleksejs Vasiļjevs
4601M
2000. gada. 18 decembrīs

Курсовой проект.

Система управления объектно-ориентированной базы данных (СУООБД).

Руководитель проекта:
Алексей Васильев

Исполнители:
Алексей Гуляев
Алексей Васильев

Содержание:

1. Терминология

СУООБД – Система управления объектно-ориентрованной базой данных.

Перманентный (persistent) объект – объект который находится в постоянной памяти, зафиксированный объект.

Временный (transient) объект – объект с которым еще работают какие либо открытые транзакции.

Нода (node) – экземпляр СУООБД.

Кластер (cluster) – совокуность нод.

ODMG – Object Database Management Group.

ODL – Object Definition Language (язык описания объектов).

OQL – Object Query Language (объектно-ориентированный язык запросов).

2. Введение

Данная курсовая работа посвящена одному из наиболее активно развивающихся направлений в последние 10 лет в области баз данных – объектно-ориентированным базам данных.

Данный проект состоит из двух частей реализованных разными людьми. В первой части описывается высокоуровневая архитектура СУООБД, во второй части – ее реализация.

Реализация данного проекта осуществлена в частичном варианте командой в составе двух человек:

1. Алексей Васильев - ответственен за физическую реализацию СУООБД с использованием языка Java (JDK 1.2.2 и выше). Пункты: 1-4, 6-8, 10.
2. Алексей Гуляев – ответственен за реализацию на данной разработанной СУООБД базы данных такой предметной области как база данных Интернет портала типа Buyer-Supplier. Пункты 5 и 9.

3. Архитектура СУООБД

3.1. Введение

Данный раздел посвящен высокоуровнему созданию системы управления объектно-ориентированной базы данных (СУООБД). Разработанная в рамках проекта концептуальная СУООБД является клиент-серверной, распределенной, многопользовательской, масштабируемой и использующей преимущества многопроцесорных систем. Масштабируемость обеспечивается посредством добавления новых серверов называемых далее узлами (ноды). Совокупность нод называется кластером.

На работу оказали влияние следующие труды в сфере ООБД:

- СУООБД O2
- СУООБД POET
- СУООБД GOODS (Generic Object Oriented Database System)
- Стандарт ODMG 3.0
- Манифест по ООБД №1 *(М. Аткинсон, Ф. Бансилон, Д. ДеВитт, К. Диттрих, Д. Майер, С. Здоник)*
- Манифест по ООБД №3 (*Х. Дарвин и К. Дэйт*)

3.2. Архитектура СУООБД

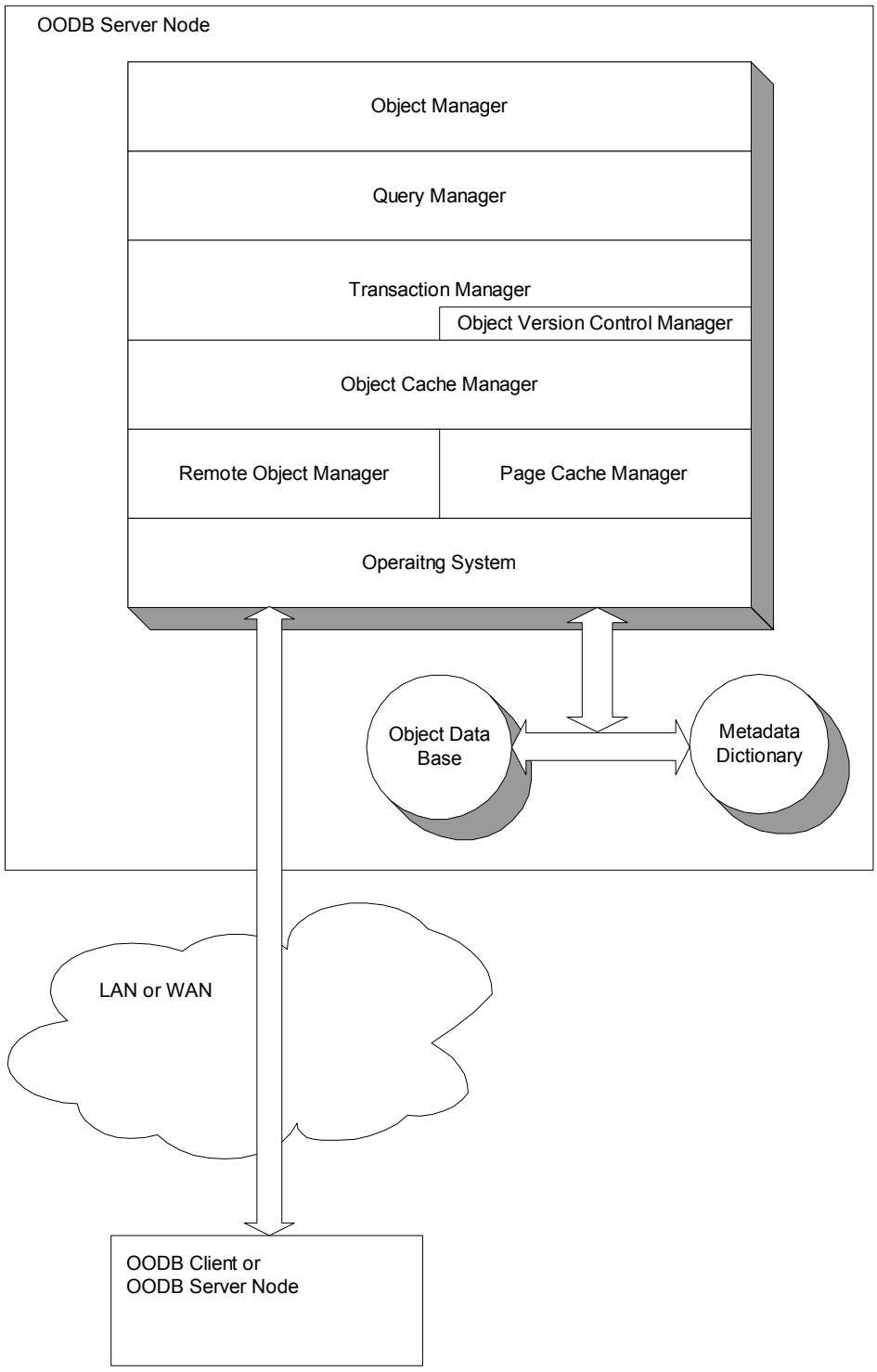


Схема 1. Архитектура системы управления ООБД.

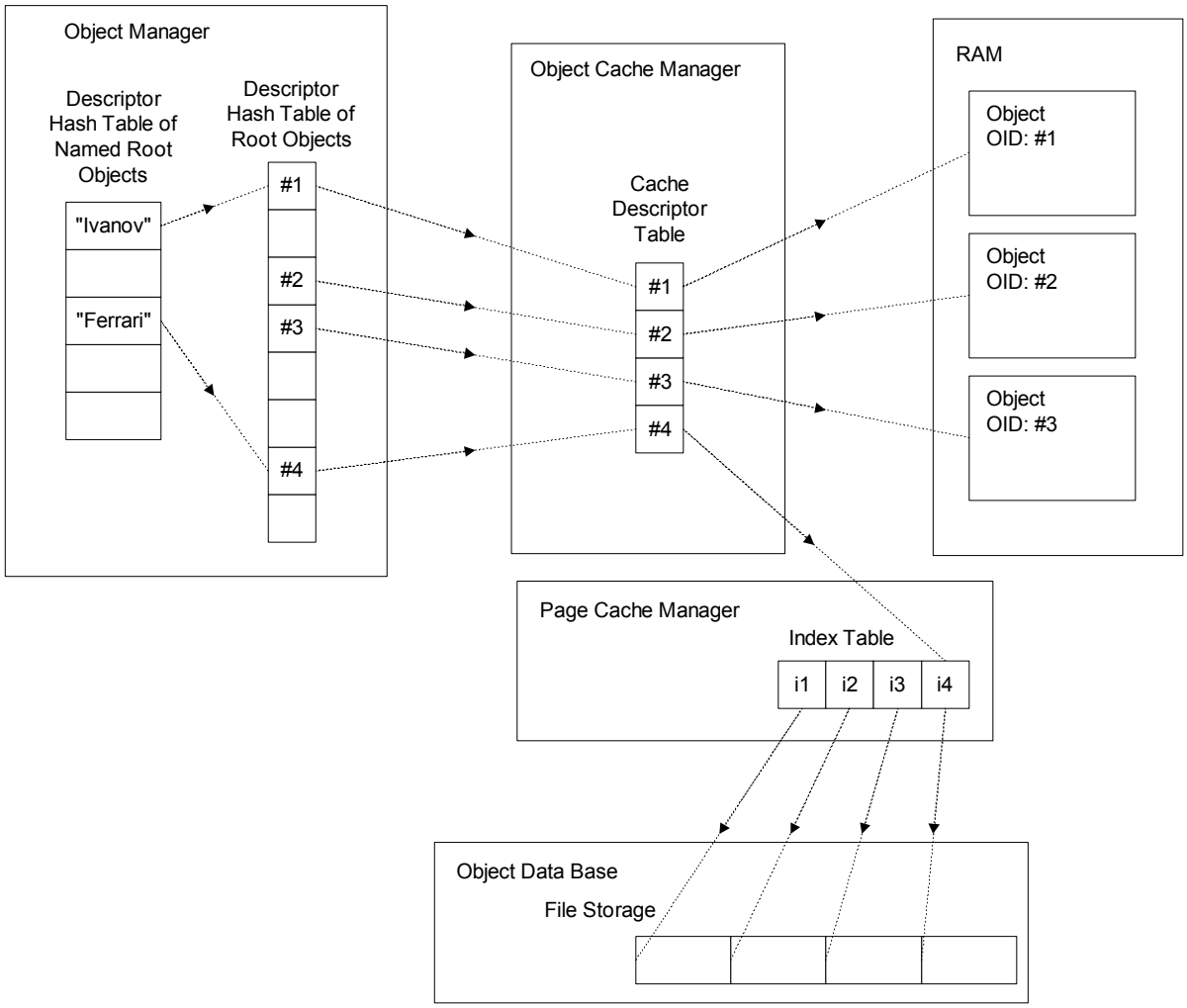


Схема 2. Структура адресации для перманентных (корневых) объектов.

3.2.1. Object Manager

Основная задача менеджера объектов – это манипуляция с объектами. Он аллоцирует и очищает память из под объектов.

Для доступа к объекту используются динамические хэш таблицы дескрипторов, которые позволяют по идентификатору объекта (OID) получить его физический адрес в памяти (mapping). Каждый объект может находится в одной из двух таблиц дескрипторов:

1. Таблица дескрипторов корневых объектов – объекты хранящиеся в базе, для которых успешно прошел транзакционный COMMIT.
2. Таблица наименовонных объектов – индексная таблица корневых объектов для доступа к корневым объектам не по OID, а по какому-нибудь имени.
3. Таблица дескрипторов временных объектов – объекты для которых не был выполнен COMMIT или которые были порождены как временные в результате транзакций и с ними еще продолжается работа.

Доступ к объектам осуществляется только посредством идентификаторов OID. Не разрешен явный внешний пользовательский запрос на получение объекта по его OID'у. Таким образом OID используется только самой СУООБД для внутренних целей как придусмотрено манифестом по ООБД №3.

3.2.1.1. OID

Менеджер объектов раздает всем вновь созданным объектам идентификаторы OID, которые уникальны в пределах одной ноды и их значения – время создания объектов в наносекундах. В кластере состоящем из нескольких распределенных нод возможна ситуация когда создадуться объекты у которых время создания совпадет. В этом случае в пределах кластера объект характеризуется составным идентификатором состоящим из имени хоста на котором он находится и своим идентификатором.

OID используется менеджером контроля версий (Object Version Control Manager) при работе с различными версиями одного и того же объекта.

3.2.1.2. Garbage Collector

Основная цель сборщика мусора – это удаление объектов из временной таблицы дескрипторов на которые больше никто не ссылается. GC – это отдельно демон запускающийся через строго определенное время (например, раз в сутки) или по определенному событию (например, при недостатке памяти) и является двух-фазным.

При его старте происходит временный останов системы и она переходит в состояние “GC_WAIT”. При этом нода которая явилась инициатором сбора мусора становится координатором и посылает запрос остальным (если есть) удаленным нодам на старт GC. Получив запрос от координатора сбор мусора нодами начинется только после того как предыдущие транзакции завершаться (переход в состояние “GC_STARTED”). После этого посылается ответ координатору. Если же хотя бы от одной ноды ответ не был получен через определенное время, то координатор посылает всем запрос на сброс GC.

Используется сборщик мусора типа “mark-and-sweep” (пометить-и-удалить). Проход по объектам начинается с корневой таблицы дескрипторов. Объекты которые уже были просканированы и ссылающиеся на другие объекты помечаются как “black”, а объекты на которые они ссылаются - как “grey”. Все остальные являются “white”. Если объект ссылается на удаленный объект, то удаленной ноде посылается соответствующий запрос.

По окончанию этого процесса все ноды посылают соответсвующее сообщение координатору. Получив их он посылает сообщение о том, что теперь они могут начать фазу чистки. На этой фазе ссылки на все “white” объекты из временной таблицы удаляются, а объекты уничтожаются.

3.2.2. Query Manager

Менеджер запросов отвечает за обработку удаленных запросы к базе данных.

Существует два подхода доступа к состоянию объекта посредством языка запросов:

1. Через методы которые возвращают или устанавливают значения атрибутов объектов. Например: STUDENT.getName() или STUDENT.setName(“IVANOV”)
2. Напрямую обращаясь к атрибутам. Например: STUDENT.name или STUDENT.name = “IVANOV” Недостаток: нарушение инкапсуляции объектов и невозможность создания вычисляемых атрибутов.

Поскольку при программировании методов класса используется процедурный язык в котором строго определена последовательность действий, то могут возникать определенные трудности при оптимизации запросов – написанных с помощью непроцедурного языка запросов.

В данной работе используется доступ к состояниям объектов только посредством вызова методов. При этом при компиляции запроса производится частичное вычисление вызываемых в нем методов, а именно методов которые ничего не делают а возвращают только состояния атрибутов, с последующим преобразованием запроса к виду, когда условия определены на атрибутах хранимых классов. После этого можно было производится обычная оптимизация запроса. Это не является нарушением инкапсуляции объектов, поскольку оптимизатор является частью системы, для которой внутренняя организация объектов БД открыта.

3.2.3. Оптимизатор запросов

Используется оптимизация запросов широко распостраненная в современных реляционнных базах данных делящийся на следующие фазы:

1. На первой фазе запрос, заданный на языке запросов, подвергается лексическому и синтаксическому анализу и компилируется во внутреннюю структуру.
2. На второй фазе запрос во внутреннем представлении подвергается логической оптимизации. Здесь используется описанный выше метод основанный на частичном вычислении вызываемых в запросе методов. После выполнения второй фазы обработки запроса его внутреннее представление остается непроцедурным, хотя и является в некотором смысле более эффективным, чем начальное.
3. Третий этап обработки запроса состоит в выборе на основе информации, которой располагает оптимизатор, набора альтернативных процедурных планов выполнения данного запроса в соответствии с его внутреннем представлением, полученным на второй фазе. Для каждого плана оценивается предполагаемая стоимость выполнения запроса. При оценках используется статистическая информация о состоянии базы данных, доступная оптимизатору. Из полученных альтернативных планов выбирается наиболее дешевый, и его внутреннее представление теперь соответствует обрабатываемому запросу.

4. На четвертом этапе по внутреннему представлению наиболее оптимального плана выполнения запроса формируется выполняемое представление плана.
5. На пятом этапе обработки запроса происходит его реальное выполнение. Это вызов интерпретатора с передачей ему для интерпретации выполняемого плана.

3.2.4. Transaction Manager

Под транзакцией понимается неделимая с точки зрения воздействия на БД последовательность операторов манипулирования данными (чтения, удаления, вставки, модификации) такая, что либо результаты всех операторов, входящих в транзакцию, отображаются в БД, либо воздействие всех этих операторов полностью отсутствует. Лозунг транзакции - "Все или ничего": при завершении транзакции оператором COMMIT результаты гарантированно фиксируются во внешней памяти (смысл слова commit - "зафиксировать" результаты транзакции); при завершении транзакции оператором ROLLBACK результаты гарантированно отсутствуют во внешней памяти (смысл слова rollback - ликвидировать результаты транзакции).

Все транзакции делятся на два типа:

1. Транзакции изменяющие схему БД. Этот режим обычно используется на стадии разработки БД,
2. Транзакции работающие непосредственно с объектами в рамках уже определенной схемы. Используется на стадии выполнения приложений.

3.2.5. Сериализация транзакций

Сериализация транзакций основана на синхронизационных захватах объектов базы данных с использованием оптимистического метода, который основывается на том, что результаты всех операций модификации базы данных сохраняются в рабочей памяти транзакций, т.е. при изменении объектов создаются сначала их копии доступные через дескриптор временных объектов. Реальная модификация объектов производится только на стадии фиксации транзакции (COMMIT). При этом ссылки на объекты перемещаются из временной таблицы дескрипторов в корневую таблицу дескрипторов.

Средства восстановления БД после сбоев и откатов транзакций также могут включаться и выключаться. Поддерживается режим, при котором все постоянно хранимые объекты загружаются в оперативную память при начале транзакции для увеличения скорости работы прикладной системы.

Подсистема управления транзакциями обеспечивает традиционную сериализуемость транзакций, а также поддерживает средства журнализации изменений и восстановления БД после сбоев. Для сериализации транзакций применяется двухфазной протокол. Журнал изменений обеспечивает откаты индивидуальных транзакций и восстановление БД после мягких сбоев (выключение питания).

Для обеспечения сериализации транзакций синхронизационные захваты объектов, произведенные по инициативе транзакции, можно снимать только при ее завершении. Это требование порождает двухфазный протокол синхронизационных захватов - 2PL. В соответствии с этим протоколом выполнение транзакции разбивается на две фазы:

1. Первая фаза транзакции - накопление захватов;
2. Вторая фаза (фиксация(COMMIT) или откат(ROLLBACK)) - освобождение захватов.

3.2.6. Блокировка объектов

Существует два типа блокировки (захвата) объектов:

1. READ (shared lock). Чтение объекта. Устанавливается когда объект зачитывается в память сервера. Возможно выполнение параллельных запросов на чтение к одному объекту.
2. UPGRADE. Изменение объекта. Устанавливается когда объект в течении транзакции изменяет свое состояние или удаляется. Невозможен параллельный доступ к объекту.
3. WRITE (exclusive lock). Создание объекта. Устанавливается когда объект в течении транзакции создается. Невозможен параллельный доступ к объекту.

3.2.7. Object Version Control Manager

Менеджер контроля версий объектов используется менеджером транзакций для обеспечения оптимистичекого измения состояния объектов.

Версии поддерживаются у всех объектов. При создании объекта его версия равна 0. При каждом обращении через экстент или по его имени происходит увеличение версии на 1.

3.2.8. Object Cache Manager

Менеджер кешей объектов отслеживает объекты к которым доступ производится наиболее часто и буферизует их в памяти. Он является промежуточным звеном между менеджером объектов и менеджером буфера страниц.

3.2.9. Page Cache Manager

Менеджер буфера страниц занимается перемещением страниц из буферов оперативной памяти во внешнюю память и наоборот, поиск и размещение объектов в буферах оперативной памяти (как принято в объектно-ориентированных системах, поддерживаются два представления объектов - дисковое и в оперативной памяти; при перемещении объекта из буфера страниц в буфер объектов и обратно представление объекта изменяется). Кроме того, эта подсистема ответственна за поддержание вспомогательных индексных структур, предназначенных для ускорения выполнения запросов. Для эффективного использования ресурсов в перспективе необходимо использовать в качестве организационной структура данных сбалансированное дерево (BTree).

Этот менеджер также ответственнен за буферизацию дисковых операций.

3.2.10. Remote Object Manager

Каждый объект в пределах кластера распределенных нод уникально идентифицируется именем ноды на которой он находится и OID'ом.

Менеджер удаленных объектов буферизирует запросы к удаленным нодам так как время передачи данных по сети относительно большое и остылает данные клиентам. В качестве клиентов могут выступать как другие удаленные ноды так и клиентские приложения.

Возможны как минимум 2 варианта реализации менеджера удаленных объектов.

1. С использованием централизованного брокера объектных запросов - Object Request Broker (ORB сервер). В этом случае выделяется отдельный узел в сети занимающийся перераспределением запросов от одной ноды к другой. В этом случае возможна стыковка с другими СУООБД использующих ODL и CORBA, так как ODL является подмножеством IDL (Interface Definition Language) используемом в CORBA серверах.

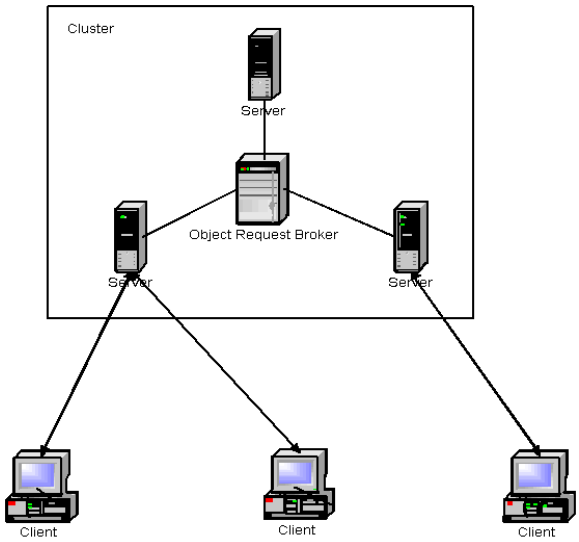


Схема 3. Архитектура кластера с использованием цетрализванного брокера запросов.

2. Без использования ORB сервера. В этом случае каждая нода имеет список всех адресов хостов нод и самостоятельно занимается адресацией.

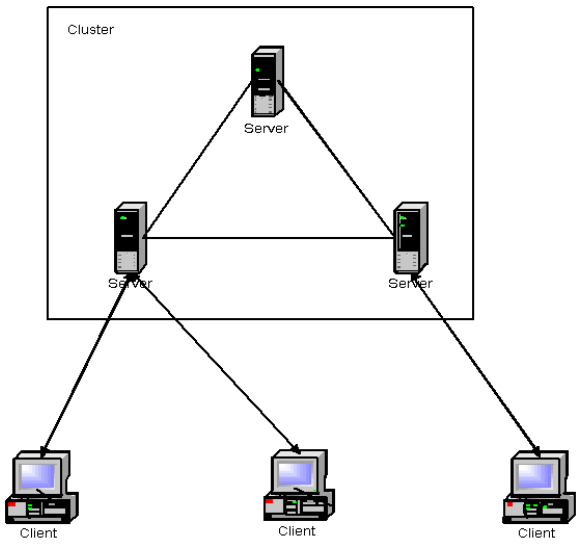


Схема 4. Архитектура кластера без использования цетрализованного брокера запросов.

3.2.11. Адресация ООБД нод

Доступ к распределенным базам осуществляется по URL'у (Unified Resource Locator) баз. Полный формат URL описывается так:

oodb://[<user>[:<password>]@]<server>[:<port>] / | <databasename>

Например "oodb://root.tsi.lv:2048/students"

3.2.12. Metadata Dictionary

Словарь метаданных (схема) - база данных о структуре объектов (описание классов). Она хранит названия, идентификаторы классов, описание атрибутов, методов и индексы описанных в классах. Для каждого класса храниться также описание предка, потомков и интерфейсов которые этот класс воплощает. При старте системы все классы автоматически зачитываются из словаря в оперативную память ООБД для последующей работы с объектами.

Словарь существует в одном экземпляре на каждой ноде и при этом на всех нодах словари одинаковы. Словарь может разделяться любым количеством баз данных объектов.

3.2.13. Object Data Base

В отличии от словаря который существует на ноде в единственном экземпляре базы данных объектов могут быть в любом количестве. Возможно задание для каждого класса своей базы данных. Это ускорит обработку запросов так как на обработку (чтение, запись) каждой базы данных создается отдельный процесс. Возможно также задание одной базы данных для всех классов, а также любой комбинации классов.

Для каждой базы данных создаются два файла: само хранилище объекотв и их индексы. Все объекты хранятся в сериализованном виде в файловых хранилищах. Сериализация объектов означает сохранение всех состояний объектов. Помимо файлов с объектами хранятся и файлы индексов.

По умолчанию объекты индексируются по их OID'ам и возможно по имени. Разрешено явное задание у классов атрибутов по которым необходимо производить индексацию.

3.2.14. OODB Client

Вся описанная выше структкура являлась описанием серверных компонент СУООБД. Помимо серверной части существет и клиентская часть.

Клиент – пользовательское приложение которое для доступа к ООБД использует специальные драйвера. Драйвера же ответственны за доставку данных и сообщений от сервера посредством специального протокола. Реализация протокола зависит от того WAN(Internet) или LAN(Intranet) сеть используется.

Для уменьшения обрашения к удаленному серверу на клиенте создается менеджер буфера объектов, аналог серверной части, который кеширует на клиенсткой стороне полученные объекты.

3.3. Создание классов

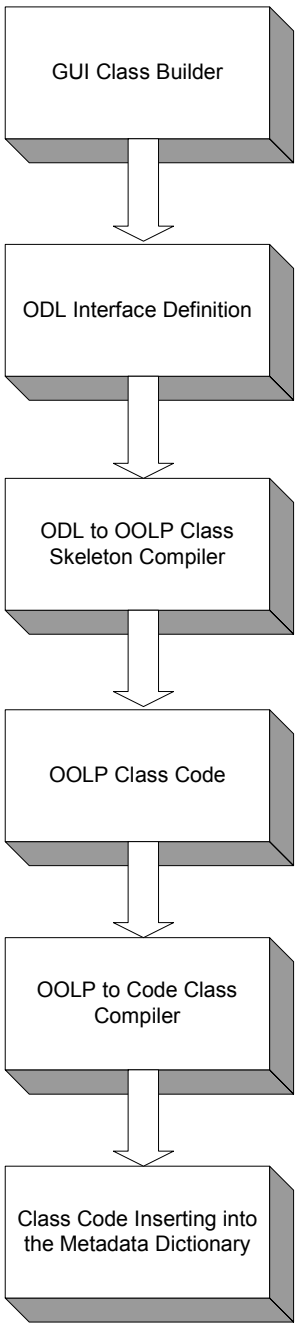


Схема 5. Структурная схема создания классов.

Конечным продуктом всего этапа создания классов является описание структуры заданной предметной области хранящейся в словаре метаданных.

3.3.1. GUI Class Builder

Графическое приложение в котором производится визуальное создание классов, отношения между классами. Создаются атрибуты и описание методов. Реализация же методов выполняется на этапе “OOLP Class Code”.

Этот Builder является вполне независимым приложением посколько выходные данные являются ODL (Object Definition Language) описанием. Это означает что его использование возможно вне зависимости от контекста данной ООБД, т.е. он может быть использован при создании классов для других СУООБД которые поддерживают язык ODL.

3.3.2. ODL Interface Definition

GUI Class Builder транслирует визуальные диаграммы в ODL (Object Definition Language) файл. Возможно также задание классов не сгенеренных с помощью Builder'а, а взятых от других СУООБД. Единственно, описания классов должны быть в ODL формате.

3.3.3. ODL to OOPL Class Skeleton Compiler

Компилирует ODL файл в "скелетный" файл объектного языка программирования для последующей физической реализации на объектно ориентированном языке програмирования (он же OML-Object manipulation language).

3.3.4. OOLP Class Code

Этот этап самый трудоемкий, поскольку включает в себя физическую реализацию описанных ранее методов на объектно-ориентированном языке програмирования. Здесь также описываются вычисляемые атрибуты.

Является частичным отображением предметной области для которой данная база данных создается.

3.3.5. OOLP to Code Class Compiler

На этом этапе производится компиляция созданного класса в двоичный код посредством стандартного компилятора языка программирования.

3.3.6. Class Code Inserting into the Metadata Dictionary

На этом этапе производится доступ на запись к словарю базы данных. Он производится посредством транзакции операции по ложению полученного двоичного кода класса в словарь метаданных. На этом этапе возможно возникновение исключительных ситуаций: например, имя класса совпадает с уже существующем или не существует заданного класса родителя в словаре. В этом случае произойдет откат транзакции с выдачей соответствующего сообщения.

4. Описание реализации СУООБД

4.1. Введение

Описанная выше архитектура реализована частично в соответствии со стандартом ODMG 3.0 (2000) с использованием связывания с языком Java (вся логика БД написанна на Java).

Данная реализация СУООБД использует одно-процессную модель с поддержкой многопотоковых транзакций (используется неявная блокировка объектов). Поддерживаются все элементарные типы данных, Java типы плюс множества, массивы, мультимножества и списки.

Java связывание обеспечивает перманентность объектов с помощью достижимости, т.е. на фиксации транзакции все объекты которые могут быть доступны из корневых объектов сохраняются в базе (становятся перманентными). Эти классы называются перманентно-совместимые классы. Они могут иметь как перманентные так и временные объекты.

Недостатки в следствии ограниченного времени выделенного на разработку:

- Отсутствует какая либо поддержка OQL запросов, менеджер удаленных объектов и сборщик мусора.
- Реализация не являетя распределенной и масштабируемой. Клиентская и серверная части совмещены, т.е. отсутствовует сетевая составляющая.
- Этап создания классов начинается сразу с этапа “OOLP Class Code”, т.е. нет визуального создания классов.
- Нельзя вводить пользовательскиие индексы. Индексация ведется только по идентификаторам объектов и именованным объектам.
- Все объекты держатся в оперативной памяти и только при закрытии БД сохраняются на диск. Более правильным здесь явилось бы использование сбалансированных деревьев BTree вместо хэш таблиц, что увеличит время доступа, но сократит ограничения на ресурсы

К достоинствам можно отнести высокое быстродействие за счет использования хэш таблиц и отсутвия межпроцессного взаимодействия: 50 тыс. операций вставки в течении одной транзакции (INSERT запрос) занимают 6 секунд (Pentium II, 233MHz) , в то время как такое же количество вставок в БД MS Access 2000 через ODBC драйвер занимает 300 секунд.

4.2. Архитектура СУООБД

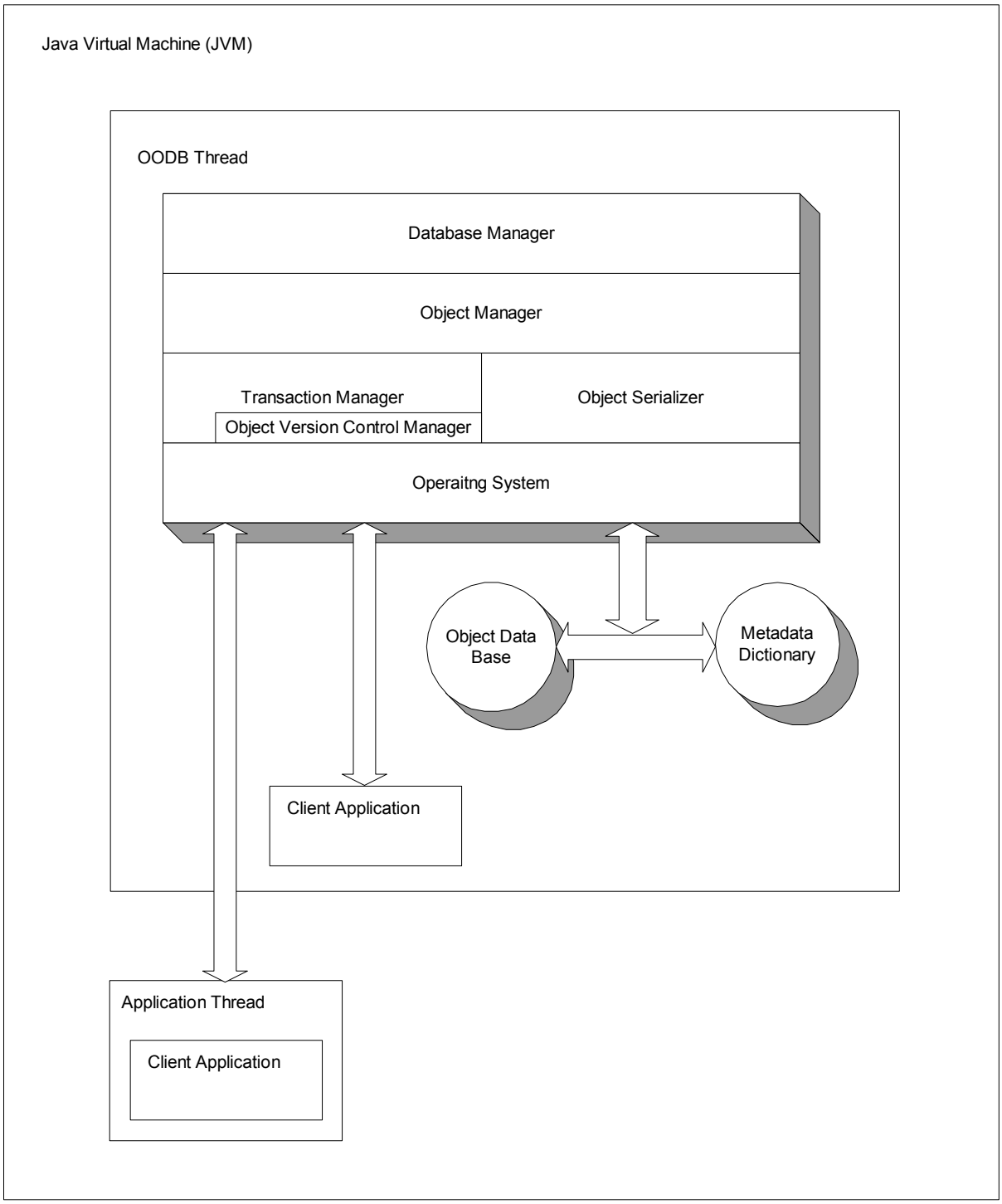


Схема 6. Архитектура реализованной системы управления ООБД.

Вся система управления объектно-ориентированной базы данных должна находиться в пределах одного адресного пространства с клиентским приложением. Возможен одновременный (конкурентный) доступ к базе из разных потоков. Все объекты базы данных должны иметь в качестве потомка системный класс MetaObject. Словарь методанных – это стандартный набор Java классов (например, jar файл) доступный из переменной окружения CLASSPATH.

Фаза существования СУООБД в памяти наступает с момента ее открытия. При этом при открытии указывается имя файла в котором находятся сериализованные объекты. Если файл не существует, то будет открыта новая пустая БД. На открытии из файла в оперативную память зачитываются все сохраненные объекты с помощью сериализатора объектов (Object Serializer). Они помещаются в хэш таблицы перманентных объектов менеджера объектов.

На фазе выполнения СУООБД обеспечивает поддержку оптимистического механизма транзакций.

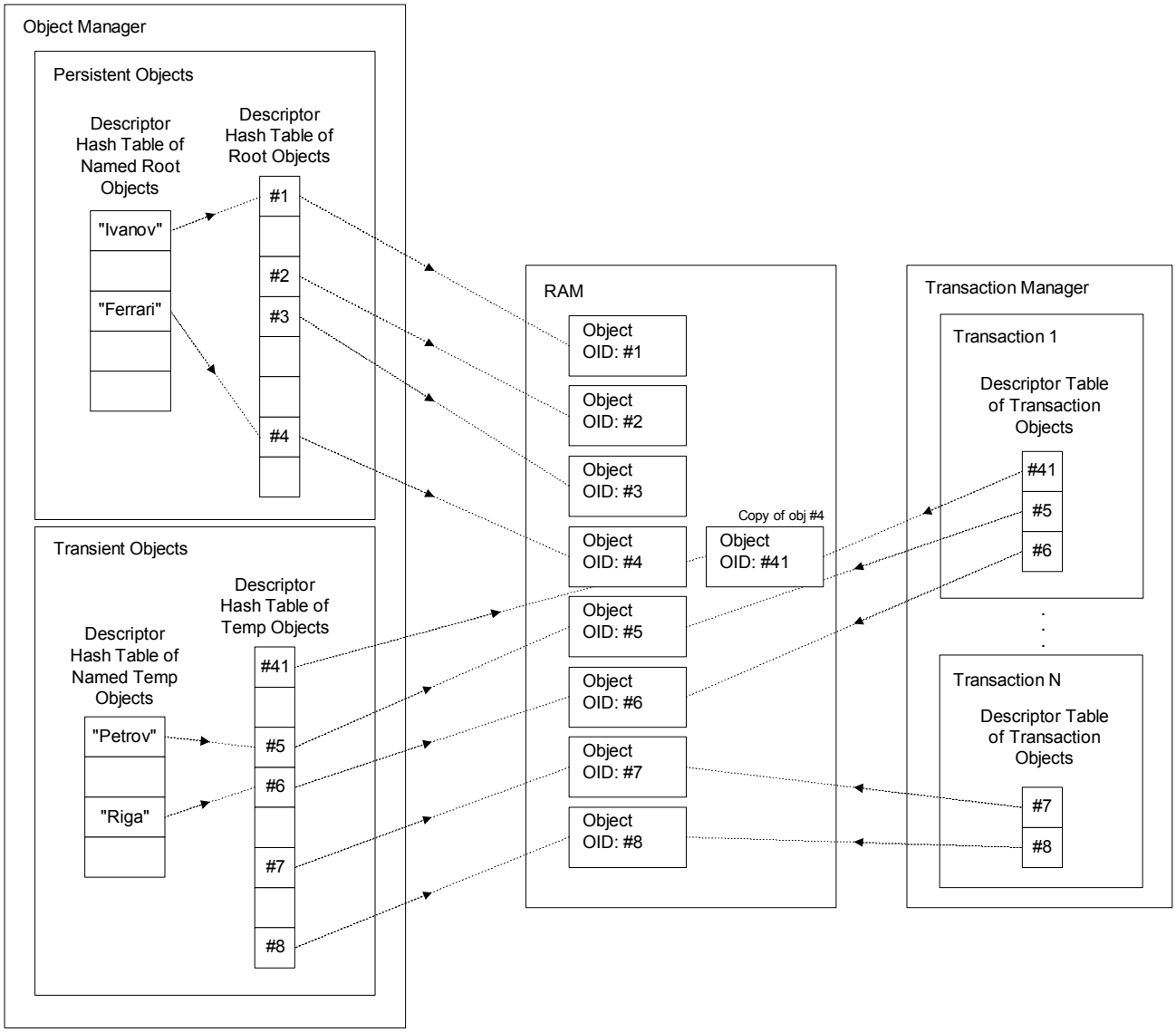


Схема 7. Структура адресации перманентных и временных объектов.

Рассмотрим для примера транзакцию № N (схема 7). При создании объекта в течении транзакции он автоматически попадает в две таблицы: в таблицу объектов данной транзакции и таблицу временных (transient) объектов менеджера объектов (объекты #7 и #8). На этапе фиксации транзакции происходит проход по таблице объектов транзакции и сверяются с объектами в таблице временных объектов у менеджера объектов. Если все в порядке (они равны), то объекты перемещаются из временной таблици в перманетную, т.е. становятся перманентными. Если в течении этого процесса возник сбой, то произойдет откат, т.е. произойдет снова проход по таблице объектов транзакции и уже зафиксированные объекты будут удалены из перманентной таблицы.

В случае с доступом (транзакция № 1) к перманентному объекту (#4) в пределах транзакции создается его копия (#41) и у оригинала увеличивается на единицу версия объекта с помощью менеджера контроля версий (Object Version Control Manager). При этом у оригинального объекта (#4) устанавливается блокировка READ, у а копии (#41) – WRITE. При фиксации транзакции эти два объекта сверяются, если они не совпадают, т.е. произошли изменения, то сверяются их версии. Если версии отличаются больше чем на единицу, то значит какая то другая транзакция успела изменить перманентный объект (#4) и произойдет откат текущей транзакции, в противном случае объект (#4) будет заменен на новый (#41).

При закрытии базы данных все перманентные объекты с помощью сериализатора объектов (Object Serializer) сохраняются на диск.

Поскольку операции открытии и закрытия базы данных очень дорогие по времени, то логично открывать базу на этапе инициалиции клиентского приложения, а закрывать - при закрытии приложения.

4.3. Транзакции

Все действия как то доступ, создание и модификация перманентных объектов должны проходить в течении открытой транзакции.

Перед выполнением любых действий над базой данных поток (thread) должен явно создать транзакционный объект или ассоциировать себя с уже существующей транзакцией (с помощью вызова, join), и эта транзакция должна быть открыта (begin). Все последующие операции потока, в том числе чтение, запись и блокировка выполняются в течении транзакции.

Поток может оперировать только с текущей транзакцией. Например, в случае попытки потока начать, зафиксировать, поставить контрольную точку (checkpoint) или откатить транзакцию (abort) предварительно не ассоцировав себя с этой транзакцией.

Транзакция может быть либо открытой, либо закрытой. Состояние транзакции можно определить с помощью вызова соответсвующего метода (isOpen).

Любой созданный в течении транзакции объект автоматически неявно получает WRITE блокировку. После успешной фиксации блокировка снимается. Если объект читается то он получает блокировку на чтение (READ), если модифицируется – на изменение (UPGRADE). Возможна явное задание блокировки объекта пользователем.

Жизненный цикл транзакции состоит из ее открытия, действий внутри ее и закрытия. Для выполнения последовательных действий в разных транзакциях возможно два подхода:

- **T1.BEGIN -> T2.COMMIT; T2.BEGIN -> T2.COMMIT;** В этом случае открывается сначала одна транзакция, затем она фиксируется и закрывается, затем открывается вторая транзакция и на фиксации закрывается.
- **T1.BEGIN -> T1.CHECKPOINT -> T1.COMMIT;** В этом случае открывается одна транзакция, затем она фиксируется без закрытия (ставиться контрольная точка) и работа происходит далее до фиксации транзакции с закрытием. Если произойдет сбой во втором участке то произойдет откат до предыдущей контрольной точки.

4.4. Типы данных

Все описанные типы данных реализованны в соответствии с типами описанными в стандарте ODMG 3.0 за исключением экстента.

4.4.1. Массив

Java класс: ArrayOfObject

ODL эквивалент: ARRAY <Type, maxSize>

Используется для предоставления отношения 1:M. Элементы массива не упорядочены и не уникальны.

4.4.2. Список

Java класс: ListOfObject

ODL эквивалент: LIST <Type>

Используется для предоставления отношения 1:M. Элементы списка упорядочены и не уникальны.

4.4.3. Мультимножество

Java класс: BagOfObject

ODL эквивалент: BAG <Type>

Предоставляет отношение 1:M. Элементы мультимножества не упорядочены и не уникальны.

Возможны следующие действия на мультимножествами: объединение, пересечение, разность, вхождение.

4.4.4. Множество

Java класс: SetOfObject

ODL эквивалент: SET <Type>

Предоставляет отношение 1:M. Элементы множества не упорядочены и уникальны.

Возможны следующие действия на множествами: объединение, пересечение, разность, определение подмножества и супермножества.

4.4.5. Экстент

Экстент (Extent) – множество всех перманентных объектов заданного класса. Его использование удобно когда необходимо вернуть все объекты одного класса и эквивалентен OQL запросу “SELECT * FROM <ClassName>”.

На текущий момент ни ODMG 3.0 ни Java не определяют структуры экстент. Однако некоторые производители СУООБД в частности (POET 6.1) используют ее.

Пример:

```
// . . .
Extent objects = new Extent("MyClass");
// . . .
```

Используя вызов методов экстента `hasNext()` и `next()` мы "проходим" по экстенту, получаем каждый элемент и выводим его на консоль.

```
// . . .
while (objects.hasNext())  {
    System.out.println(objects.next());
}
```

4.5. Запросы к базе данных

Все запросы к базе данных могут быть реализованы только с использованием языка связывания, т.е. Java, поскольку не поддреживается язык запросов OQL.

Запрашивать объекты можно как с помощью структуры Extent, так и с помощью именованных объектов.

4.6. Пример реализации базы данных

Рассмотрим пример реализации базы данных. Пусть имеется класс студент и класс преподаватель связанные отношением 1:M.

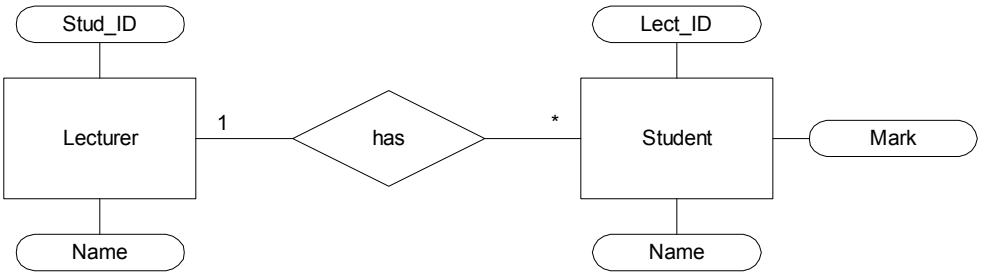


Схема 8. Реляционная модель сущность-связь для БД студент-преподаватель.

Пусть имеется реляционная модель. Преобразуем ее в объектно-ориентированную UML модель.

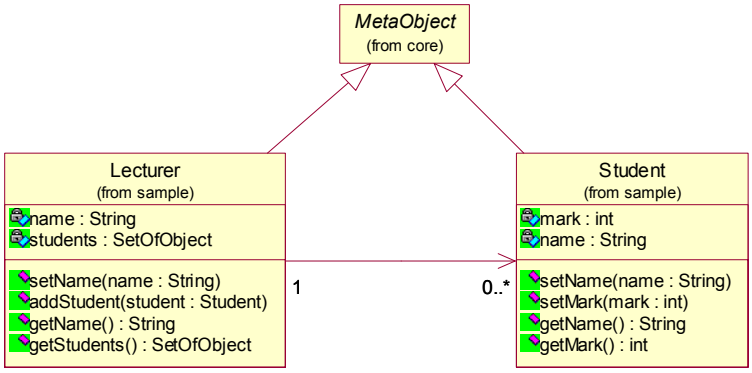


Схема 9. Диаграмма классов для БД студент-преподаватель.

Главное отличие от реляционной модели – это наличие у класса Lecturer множества в котором и находятся экземпляры класса Student. Реализуем объектную модель в Java.

4.6.1. Класс Lecturer

```
import lv.tsi.oodb.odmg.SetOfObject;
import lv.tsi.oodb.core.MetaObject;

/**
 * Lecturer class definition.
 */
public class Lecturer extends MetaObject {

    private SetOfObject students = new SetOfObject();
    private String name;

    public Lecturer() {}

    public void addStudent( Student stud ) {
        students.add( stud );
    }

    public void setName( String name ) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public SetOfObject getStudents() {
        return students;
    }

}
```

4.6.2. Класс Student

```
import lv.tsi.oodb.core.MetaObject;

/**
 * Student class definition.
 */
public class Student extends MetaObject {

    private String name;
    private int    mark;

    public Student() {}

    public void setName( String name ) {
        this.name = name;
    }

    public String getName() {
        return name;
    }

    public void setMark(int mark) {
        this.mark = mark;
    }

    public int getMark() {
        return mark;
    }

}
```

4.6.3. Запросы к БД

Рассмотрим два запроса выполенные в Java реализации.

1. Создать одного преподавателя, двух студентов и связывать их друг с другом.
2. Получить преподователя по его имени и подсчитать среднюю оценку по всем его студентам.

4.6.3.1. Запрос на добавление одного преподователя (Ulman) и двух студентов (Ivanov, Petrov)

```
import org.odmg.*;
import lv.tsi.oodb.odmg.*;

public class Insert {

    public static void main( String[] str ) throws Exception {
        Database db = new Database();
        //  ------------------------------------------------
        // | Opening the database
        //  ------------------------------------------------
        try {
            db.open( "student", Database.OPEN_READ_WRITE );
        } catch (ODMGException e) {
            System.out.println( "Error." );
             System.exit(1);
        }
        //  ------------------------------------------------
        // | Binding object to the database
        //  ------------------------------------------------
        Transaction txn = new Transaction( db );
        txn.begin();

////////////////////////////////////////////////
////// The MAIN PART  /////////////////////////
////////////////////////////////////////////////
        try {
            Lecturer p = new Lecturer();
             p.setName("Ulman");

             Student s = new Student()
              s.setName("Ivanov");
              s.setMark(3);

             p.addStudent( s );

            s = new Student()
              s.setName("Petrov");
              s.setMark(5);

             p.addStudent( s );
            db.bind( p, " Ulman " );
        } catch (Exception e) {
            txn.abort();
            throw e;
        }
////////////////////////////////////////////////
        txn.commit();
        //  ------------------------------------------------
        // | Closing the database
        //  ------------------------------------------------
        try {
           db.close();
        } catch (ODMGException e) {
            System.out.println( "Error." );
        }

      }
}
```

4.6.3.2. Подсчет средних оценок у всех студентов проподавателя “Ulman”

```
import org.odmg.*;
import lv.tsi.oodb.odmg.*;

public class Select {

    public static void main( String[] str ) throws Exception {
        Database db = new Database();
        //  ------------------------------------------------
        // | Opening the database
        //  ------------------------------------------------
        try {
            db.open( "student", Database.OPEN_READ_WRITE );
        } catch (ODMGException e) {
            System.out.println( "Error." );
             System.exit(1);
        }
        //  ------------------------------------------------
        // | Getting object from the database
```

```
        //  -----------------------------------------------
        Transaction txn = new Transaction( db );
        txn.begin();

////////////////////////////////////////////////
////// The MAIN PART  /////////////////////////
////////////////////////////////////////////////
        try {
            Lecturer p = (Lecturer)db.lookup("Ulman");

             Iterator i = p.getStudents().iterator();
              int total = 0;
              int count = 0;
             while (i.hasNext()) {
                 total += ((Student)i.next()).getMark();
                 count++;
             }
             System.out.println( "Avg: " + total/count );
         } catch (Exception e) {
             txn.abort();
             throw e;
        }
////////////////////////////////////////////////
        txn.commit();
        //  -----------------------------------------------
        // | Closing the database
        //  -----------------------------------------------
        try {
           db.close();
        } catch (ODMGException e) {
            System.out.println( "Error." );
        }

      }
```

5. Объектно-ориентированная база данных Buyer-Supplier

5.1. Введение

Данная работа посвящена созданию объектно-ориентированной базы данных (ООБД). Разработанная в среде СУООБД данная база представляет собой 100% объектно-ориентированную базу данных, которая полностью соответствует стандарту ODMG. Данная база разрабатывалась согласно общей схеме разработки подобных проектов – т.е. идеи → ODL → реальное воплощение. В качестве среды реализации был использован объектно-ориентированный язык Java (версия JDK 1.2.2 и выше). В качестве СУООБД был взят разработанный А.В. вышеизложенный проект СУООБД.

В качестве предметной области была выбрана тема – Схема обработки заказов Интернет магазина. При разработки основных бизнес объектов в качестве отправной точки была использована схема реально существующей реляционной базы данных.

5.2. Описание логической схемы разрабатываемой БД.

Данная БД представляет собой ордерную часть большой схемы, предназначенной для функционирования электронного магазина типа Buyer-Supplier. В данной схеме присутствуют следующие бизнес-объекты:

1. Ордер(Order) – представляет собой запрос на приобретение набора товаров разных поставщиков одним покупателем. По сути представляет собой аналог каталога файловой системы, т.к. реально содержит не товары, а ссылки на субордера(см. Пункт 2).
2. Субордер(SubOrder) – главный бизнес – объект (БО) системы вокруг которого и происходят реальные действия. Содержит некоторое количество разных продуктов от одного поставщика. Может принимать статус от New → Shipped. Содержит ссылку на Поставщика, Покупателя, Доставщика. Реально ссылается на набор Item'ов.
3. Item(Элемент) – Бизнес-объект представляющий некоторое количество одного продукта. Имеет статус. Обязательно ссылается на соответствующий субордер.
4. Item Detail(Описание Элемента) - Бизнес – объект представляющий набор действий произведенных над неким Item'ом. По сути – это лог список статусов элемента субордера. Предполагается, что он может находится в следующих статусах: Shipped, Cancelled, Returned, In Process. Смысл статусов говорит сам за себя.
5. Delivery(Фирма - доставка) – БО – фирма отвечающая за доставку субордера. Связана с каждым заказом.
6. Organization(Организация) – БО – организация поставщиков или покупателей. Каждый пользователь ссылается на определенную организацию (является ее сотрудником).
7. User(Пользователь) – Поставщик или Покупатель в системе. Связан с организацией. Кроме этих 2-х типов может принимать и специфические типы(например "Главный покупатель").
8. Product(Продукт) – БО, который представляет собой объект сделки. Принадлежит определенной фирме – производителю. Имеет определенный набор атрибутов(как то вес, размеры, упаковка и т.д.)
9. Manufacturer(Производитель) – БО объект – производитель товаров. Имеет ссылки на все товары относящиеся к его производству.

5.3. Построение Use Case диаграммы системы.

Воспользуемся таким инструментом из общего набора ОО – средств, как Use Case диаграмма, чтобы более детально прояснить принцип работы системы – и на основании этой схемы уже разработать объекты ODL и связи между ними.

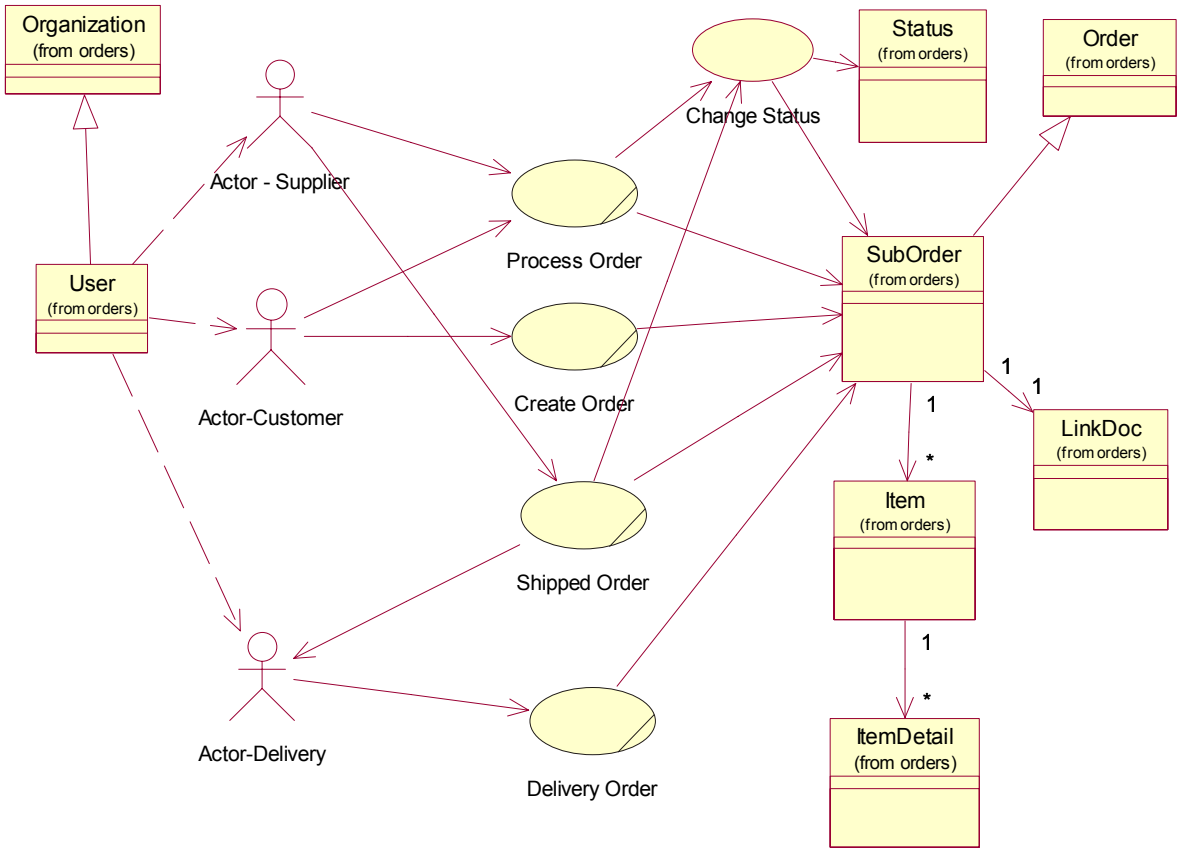


Схема 10. Use Case диаграмма системы

Основываясь на этой схеме перейдем к описанию объектов ODL (см. пункт 5.4).

5.4. Разработка и описание объектов модели ODL

Некоторые пояснения относительно нижепреведенных классов:

- *interface – представляет собой ключевое слово – обозначающее, что описывается класс*
- *attribute – указывает на свойство класса (атрибут).*
- *relationship – указывает на связь с другим классом(1→1, 1→M,M→1,M→M).Соответственно применяется ключевое слово Set.*
- *inverse – указывает, что существует связь в обратном направлении.*

Количество классов описанных ниже превосходит количество БО описанных в пункте 5.2. Это результат удаления избыточности в проекте нашей БД.

1) *interface* **Order**{
attribute String number;
attribute String type;
attribute Date createDate;
attribute Date updDate;

```
    relationship Set <SubOrder> subOrders
                inverse SubOrder::order;
    relatonship Status status;
    relationship Set <User> users;
  }
```

2) *interface* Sub**Order** : **Order**{

```
    attribute String pon;
    attribute Float totalSum;
    attribute Float totalDiscount;
    attribute Float dlvPrice;
    attribute String shipperName;
    attribute String paymentObjID;
    attribute String paymentClassID;

    relationship Order order
                inverse Order::subOrders;
    relationship Set <Item> items;
    relationship LinkDoc linkDoc;
    relationship Delivery delivery;
  }
```

3) *interface* **Item**{

```
    attribute Integer quantity;
    attribute String parameter;
    attribute Float price;
    attribute Float discount;
    attribute String description="";
    attribute Date updDate;
    attribute String sku;
    attribute Float srcPrice;

    relationship SubOrder suborder
                inverse SubOrder::items ;
    relationship Product product;
    relationship User updater;
    relationship Status status;
    relationship Set<ItemDatail> itemDetails;

  }
```

4) *interface* **ItemDetail**{

```
    attribute int qty;
    attribute Date createDate;
    attribute String description;
    attribute boolean undone;

    relationship Status status;
    relationship Item item
     inverse Item::itemss ;
```

relationship User user;

}

5) *interface* **Product**{

attribute String class_id;
attribute String upc;
attribute String model;
attribute Float map_price;
attribute Float width;
attribute Float length;
attribute Float height;
attribute Float weight;
attribute Float length_package;
attribute Float height_package;
attribute Float weight_package;
attribute Date updDate;

relationship Set<Categories> categories;
relationship Manufacturer manufacturer
inverse Set<Manufacturer>:: products;
relationship User supplier;
relationship Item item
inverse Item::product;

}

6) *interface* **Manufacturer**{

attribute String classID;
attribute String logo;
attribute String name;
attribute String code;

relationship Set<Product> products
inverse Product::manufacture;
relationship Address address;

}

7) *interface* **Category**{

attribute String classID;
attribute String name;
attribute String identifier;
attribute String howToUse;
attribute String description;

relationship Set<Product> products
inverse Product::category;

}

8) *interface* **Status**{

```
        attribute char statusValue;
        attribute String statusName;
        attribute int statusWeight;
        attribute String statusClassID;
        attribute String description;

}
```

9) *interface* **LinkDoc**{

```
        attribute String docNum;
        attribute String notes;
        attribute Date lastDate;
        attribute String docType;

        relationship SubOrder subOrder
                inverse SubOrder::linkDoc;
}
```

10) *interface* **Delivery**{

```
        attribute String name;
        attribute String description;
        attribute Date updDate;
        attribute String trackingUrl;

        relationship SubOrder subOrder
                inverse SubOrder::delivery;
        relationship Set<Truck> trucks
                inverse Delivery::delivery;
       relationship Address address;
}
```

11) *interface* **Truck**{

```
       attribute String truckNumber;
       attribute Date dlvDateFact;
       attribute Date createDate;
       attribute String truckNotes;

        relationship Delivery delivery
                inverse Set<Delivery>::trucks;
}
```

12) *interface* **User**{

```
       attribute String classID;
       attribute String name;
       attribute String language;
       attribute String orgClass;
       attribute String department;
       attribute Date dateReg;
```

```
        attribute boolean valid;

        relationship Organization organization;

}
```

13) *interface* **Organization**{

```
        attribute String businessType;
        attribute String classID;
        attribute boolean valid;
        attribute Date dateReg;

        relationship Address address;
        relationship Set<User> employees
                inverse User::organization;
}
```

14) *interface* **Address**{

```
        attribute String classID;
        attribute String street;
        attribute String street2="";
        attribute String city;
        attribute String state;
        attribute String country;
        attribute String zip;
        attribute String www="";
        attribute String receiverNotice="";
        attribute String description="";
        attribute String email="";
        attribute String phones="";
        attribute String faxes="";
}
```

5.5. Связи в ООБД (модель ER)

Чтобы наиболее понятно представить себе набор связей в нашей ООБД, воспользуемся схемой сущность-связь (модель ER), которая хотя и применяется для проектирования реляционных БД, но исключительно удобна в нашем случае. Примем наши объекты за сущности, а атрибуты relation за связи. А далее пользуясь известной схемой построим ER-модель:

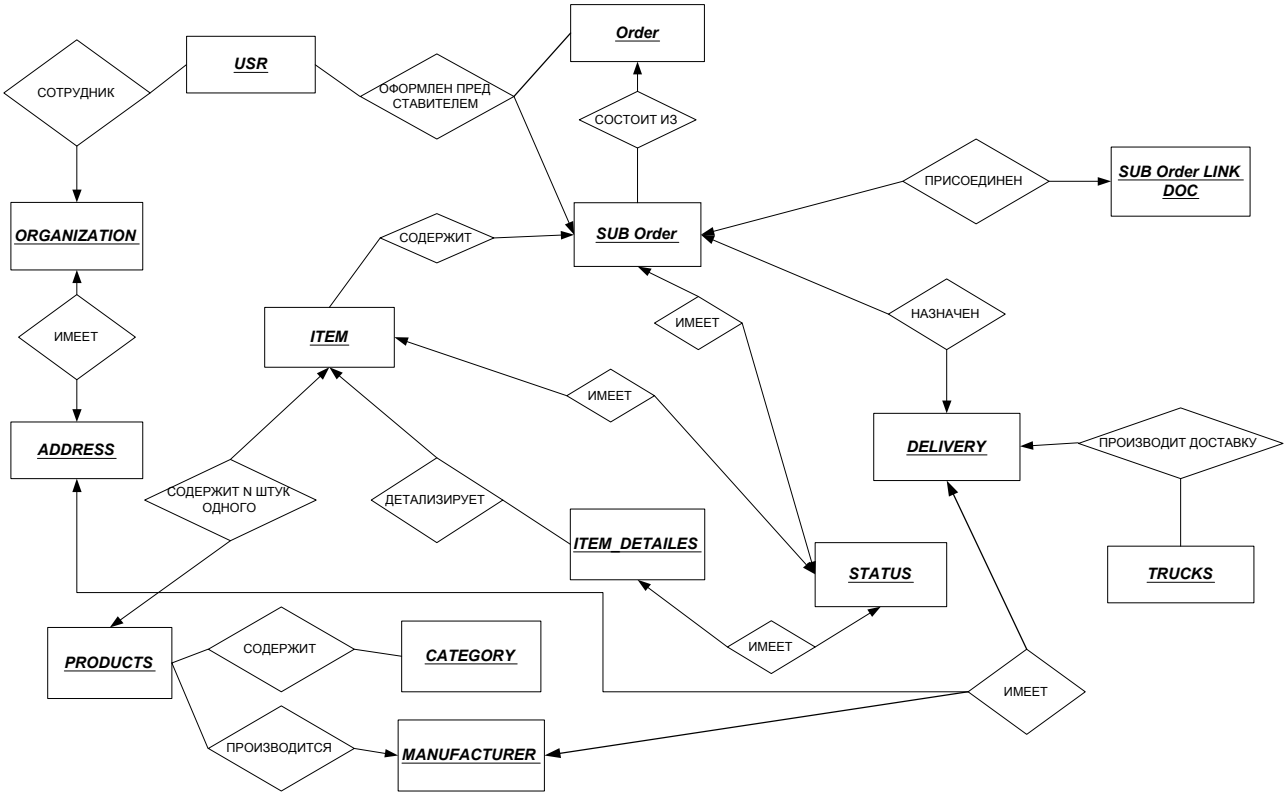


Схема 11. ER модель ООБД “Orders”.

5.6. Связывание с языком программирования (Java)

Связывание СУОБД с языками программирования базируется на следующих принципах.

1. Существует единая унифицированная система типов для языка программирования и базы данных. Экземпляры этих типов являются либо временно, либо постоянно хранимыми.
2. Связывание ODL с конкретным языком программирования должно удовлетворять синтаксису и семантике этого языка программирования.
3. Связывание реализуется посредством небольших расширений языка программирования.
4. Должна существовать возможность образования произвольных композиций выражений манипулирования данными и выражений языка программирования.

Связывание ODL с Java заключается в отображении объектной модели в Java введением совокупности классов, которые могут иметь как постоянно, так и временно хранимые экземпляры. Такие классы называются классами базы данных. Эти классы отличаются от обычных классов Java, содержащих лишь временно хранимые экземпляры. Для каждого класса базы данных X образуется вспомогательный класс указателей Ref<X>. В действительности Ref<X> эквивалентен указателю (X*), благодаря чему манипулирование этими указателями идентично.

Стандарт ODMG 1.2 (1993) определяет отображение типов ODL в типы C++, образуя расширенный язык C++ ODL. Для манипулирования объектами предоставляется библиотека классов C++ и функций, реализующая манипулирование базой данных, поддержку классов C++, эквивалентных встроенным типам ODL (например типам коллекций, связей), поддержку транзакций, динамический вызов запросов на OQL и др.

В нашем случае поскольку отображение идет не в C++, а в Java пользуемся не стандартным библиотеками классов C++ (хотя возможен и такой вариант) – а специальным набором классов из нашей СУООБД. Данный набор классов Java обеспечивает тот же набор функций, что и выше перечисленный, за исключением некоторых модификаций (за более подробной информацией см. вторую часть описания посвященную ядру)

По сравнению с описанием ODL в описание каждого класса на Java добавилось описание набора методов:

- Методы отвечающие за инициализацию экземпляра (т. н. констструкторы):

```
public void init(String number, String type, Date createDate, Date updDate ) {
   this.number      = number;
   this.type        = type;
   this.createDate  = createDate;
   this.updDate     = updDate;
}
```

- Методы отвечающие за установление значения атрибута (set-методы):

```
public void setType(String type) {
   this.type        = type;
}
```

- Методы отвечающие за получение значения атрибута (get-методы):

```
public String getType(){
   return type;
}
```

- Методы отвечающие за установление связи с другим объектом (связь 1→1;1→M;M→M):

```
public void addSubOrder( SubOrder sb ) {
   subOrders.add( sb );
}

public void linkToStatus( Status stat ) {
   status = stat;
}
```

- Методы отвечающие за получение связанного с ним объекта (набора объектов) :

```
public Status getStatus() {
    return status;
}

public SetOfObject getSubOrders() {
    return subOrders;
}
```

5.7. Пример функционирования ООБД

Для начала приведем пример класса, создающего набор записей в нашей базе:

```
package lv.tsi.oodb.db.query;
import lv.tsi.oodb.db.orders.*;
import lv.tsi.oodb.odmg.*;
import org.odmg.ODMGException;
import java.util.*;

/**
 * ODMG classes implementation test.
 *
 * @author  Alexey Vasilyev
 * @author  Alexey Gulyaev
 * @version 0.0.3 16 December 2000
 */

public class Init_DB {
////////////////////////////////////
private static String DATABASE_NAME = "Buyer2Supplier";
```

```
public static void main( String[] str ) throws Exception {
    Database db = new Database();
    // ------------------------------------------------
    // | Opening the database
    // ------------------------------------------------
    try {
            System.out.print( "Opening database..." );
       db.open( DATABASE_NAME, Database.OPEN_READ_WRITE );
            System.out.println( "ok." );
    } catch (ODMGException e) {
       System.out.println( "error.\nExited." );
            System.out.print( e.getMessage() );
            System.exit(1);
    }
    // ------------------------------------------------
    // | Binding object to the database
    // ------------------------------------------------
    Transaction txn = new Transaction( db );
    txn.begin();
    long time = System.currentTimeMillis();

////////////////////////////////////////////
////// The MAIN PART  ////////////
////////////////////////////////////////////
    try {

      // 1. Create some Statuses
        Status s_new = new Status();
        s_new.init("New", "Order");
        Status s_receiv = new Status();
        s_receiv.init("Received", "Order");
      //----------------------
      // 2. Create some Org Addresses
        Address a_buyer = new Address();
        a_buyer.init( "OBuyer", "Lomonosova 1", "", "Riga", "", "Latvia", "LV-1019");

        Address a_supp = new Address();
        a_supp.init( "OCustomer", "Terbatas 3", "", "Riga", "", "Latvia", "LV-1023");

        Address a_dlv = new Address();
        a_dlv.init( "Delivery", "123 street", "", "NeyYork", "NY", "USA", "LV-1023");
      //----------------------
      // 3. Create one Delivery object
        Delivery d1 = new Delivery();
        d1.init( "FedEx", new Date(), "www.fedex.com");
      //------------------------------------------------
        // 3.1 Link Delivery to Address
          d1.linkToAddress(a_dlv);
        //--------------------------------
      // 4. Create some Organization
        Organization org_b1 = new Organization();
        org_b1.init("Udod", "Home Electronics", "E-Tailer");

        Organization org_b2 = new Organization();
        org_b2.init("Veniki", "Home Electronics", "E-Tailer");

        Organization org_s1 = new Organization();
        org_s1.init("Mols2", "Home Electronics", "E-Customer");
      //--------------------------------------------------------
        // 4.1 Link Organizations to Addresses
          org_b1.linkToAddress(a_buyer);
          org_s1.linkToAddress(a_supp);
```

```
   //--------------------------------
// 5. Create some Users (buyers & suppliers)
   User u_b1 = new User();
   u_b1.init("BuyerUser", "Arnold Z", "LV", "Market" );

   User u_b2 = new User();
   u_b1.init("MBuyerUser", "Vasilij V", "RU", "E-Market" );

   User u_s1 = new User();
   u_s1.init("MSupplier", "Robert M", "EN", "E-Custom");
//-----------------------------------------------------
   // 5.1 Link Organizations to Addresses
      org_b1.addEmployee(u_b1);
      org_b1.addEmployee(u_b2);
      org_s1.addEmployee(u_s1);
   //--------------------------------
// 6. Create some Manufacturers
   Manufacturer m1 = new Manufacturer();
   m1.init( "FManufacture", "Siemens", "123" );

   Manufacturer m2 = new Manufacturer();
   m2.init("ZManufacturer", "Bosch", "3443" );
//-----------------------------------------------------
   // 6.1 Link Organizations to Addresses
    //  m1.linkToAddress();
    //  m2.linkToAddress();
   //--------------------------------
// 7. Create some Products
   Product p1 = new Product();
   p1.init("ProductM", "S45", "SIEMENS S45", 300, 10, 10, 0.2F, 0.2F , 10, 0.4F, 15);
   Product p2 = new Product();
   p2.init("ProductM", "S35", "SIEMENS S25", 200, 10, 10, 0.2F, 0.2F , 10, 0.4F, 15);
   Product p3 = new Product();
   p3.init("ProductM", "S25", "SIEMENS S25", 150, 10, 10, 0.2F, 0.2F , 10, 0.4F, 15);
   Product p4 = new Product();
   p4.init("ProductM", "C35", "SIEMENS C35",  200, 10, 10, 0.2F, 0.2F , 10, 0.4F, 15);
   Product p5 = new Product();
   p5.init("ProductM", "WFL-2000", "BOSCH WASHER WFL-2000",  700, 10, 10, 0.2F, 0.2F , 10, 0.4F, 15);
//-----------------------------------------------------

  // 7.1 Link Manufacturers to Products
     m1.addProduct(p1);
     m1.addProduct(p2);
     m1.addProduct(p3);
     m1.addProduct(p4);
     m2.addProduct(p5);
 //--------------------------------

 Order o = new Order();
 o.init( "8321-32423", "WEB" );

 o.addUser(u_b1);
 o.addUser(u_s1);

   SubOrder s1 = new SubOrder();
   s1.init("8392-3819", "INVENTORY", "21", 0.5F,  "", "0x930232", "0x394222");
   s1.addUser(u_b1);
   s1.addUser(u_s1);

     Item i1 = new Item();
     i1.init(10, null, 300, 0.4F, "Telefon SIEMENS S45", 300);
     i1.linkToProduct(p1);
```

```
            Item i2 = new Item();
            i2.init( 1, null, 200, 0.4F, "Telefon SIEMENS S35", 200);
            i2.linkToProduct(p2);

            Item i3 = new Item();
            i3.init( 2, null, 180, 0.4F, "Telefon SIEMENS S25", 180);
            i3.linkToProduct(p3);

            Item i4 = new Item();
            i4.init( 4, null, 160, 0.4F, "Telefon SIEMENS C35", 160);
            i4.linkToProduct(p4);

          s1.addItem( i1 );
          s1.addItem( i2 );
          s1.addItem( i3 );
          s1.addItem( i4 );
          s1.linkToStatus(s_new);
          s1.linkToDelivery(d1);

          SubOrder s2 = new SubOrder();
          s2.init("8395-3812", "INVENTORY", "21", 0.43F, "", "0x930232", "0x394222");
          s2.addUser(u_b1);
          s2.addUser(u_s1);

            i1 = new Item();
            i1.init( 15, null, 500, 0.4F, "Bosh Washer ", 500);
            i1.linkToProduct(p5);
          s2.addItem( i1 );
          s2.linkToStatus(s_new);
          s2.linkToDelivery(d1);
        o.addSubOrder( s1 );
        o.addSubOrder( s2 );
        o.linkToStatus(s_new);

      } catch (Exception exc) {
        // in case something went wrong, we abort the transaction
        // this is especially important to release all locks
        // then we "rethrow" the exception
        txn.abort();
        throw exc;
      }
////////////////////////////////////////////
      txn.commit();
      System.out.println("Query time: " + (System.currentTimeMillis() - time)/1000 + " sec." );
      txn.dump();
      db.dump();
      // ------------------------------------------------
      // | Closing the database
      // ------------------------------------------------
      try {
            System.out.print( "Closing database..." );
            db.close();
            System.out.println( "ok." );
      } catch (ODMGException e) {
            System.out.println( "error.\nExited." );
            System.out.print( e.getMessage() );
      }
      }
}
```

Приведем несколько классов – примеров обращения к базе, аналогичных OQL (текущая реализация СУООБД не поддерживает стандарта OQL – вместо этого используется обращение к классу Extent – который возвращает все экземпляры запрашиваемого класса – а затем можно осуществить поиск по полученному множеству):

- Подсчет суммы заказа для одного СубОрдера (идет обращение ко всем его элементам)

```
 package lv.tsi.oodb.db.query;
import lv.tsi.oodb.core.Extent;
import lv.tsi.oodb.db.orders.*;
import lv.tsi.oodb.odmg.*;
import org.odmg.ODMGException;
import java.util.*;

/**
 * ODMG classes implementation test.
 *
 * @author  Alexey Vasilyev
 * @author  Alexey Gulyaev
 * @version 0.0.1 10 November 2000
 */
public class SelectTotalSubOrderSum {


////////////////////////////////////
//
   private static String DATABASE_NAME = "Buyer2Supplier";

   public static void main( String[] str ) throws Exception {
      Database db = new Database();
      // ------------------------------------------------
      // | Opening the database
      // ------------------------------------------------
      try {
            System.out.print( "Opening database..." );
         db.open( DATABASE_NAME, Database.OPEN_READ_WRITE );
            System.out.println( "ok." );
      } catch (ODMGException e) {
         System.out.println( "error.\nExited." );
            System.out.print( e.getMessage() );
            System.exit(1);
      }
      // ------------------------------------------------
      // | Binding object to the database
      // ------------------------------------------------
      Transaction txn = new Transaction( db );
      txn.begin();

      long time = System.currentTimeMillis();

//////////////////////////////////////////////
////// The MAIN PART  ///////////////////////
//////////////////////////////////////////////
      try {

         Extent ext = new Extent("lv.tsi.oodb.db.orders.SubOrder");

         float sum = 0;
         while (ext.hasNext())  {
            SubOrder s = (SubOrder)ext.next();
            sum += s.getTotalSum();
         }
         System.out.println( "Total sum:  " + sum + " $");
```

```
        } catch (Exception exc) {
            // in case something went wrong, we abort the transaction
            // this is especially important to release all locks
            // then we "rethrow" the exception
            txn.abort();
            throw exc;
        }
////////////////////////////////////////////
////////////////////////////////////////////
        txn.commit();
        System.out.println("Query time: " + (System.currentTimeMillis() - time)/1000 + " sec." );
        txn.dump();
        db.dump();

        // ------------------------------------------------
        // | Closing the database
        // ------------------------------------------------
        try {
                System.out.print( "Closing database..." );
                db.close();
                System.out.println( "ok." );
        } catch (ODMGException e) {
                System.out.println( "error.\nExited." );
                System.out.print( e.getMessage() );
        }

    }

}
```

- Пример класса осуществляющего добавление записей в базу используя bind-метод (см. описание СУООБД):

```
package lv.tsi.oodb.db.query;

import lv.tsi.oodb.db.orders.*;
import lv.tsi.oodb.odmg.*;
import lv.tsi.oodb.core.*;

import org.odmg.ODMGException;

import java.util.*;

/**
 * ODMG classes implementation test.
  *
 * @author  Alexey Vasilyev
 * @author  Alexey Gulyaev
* @version 0.0.1 10 November 2000
 */
public class InsertNamedSubOrder_Items {


////////////////////////////////////
//
    private static String DATABASE_NAME = "Buyer2Supplier";

    public static void main( String[] str ) throws Exception {
        Database db = new Database();
```

```
        // ------------------------------------------------
        // | Opening the database
        // ------------------------------------------------
        try {
                System.out.print( "Opening database..." );
           db.open( DATABASE_NAME, Database.OPEN_READ_WRITE );
                System.out.println( "ok." );
        } catch (ODMGException e) {
           System.out.println( "error.\nExited." );
                System.out.print( e.getMessage() );
                System.exit(1);
        }
        // ------------------------------------------------
        // | Binding object to the database
        // ------------------------------------------------
        Transaction txn = new Transaction( db );
        txn.begin();

        long time = System.currentTimeMillis();

////////////////////////////////////////////
////// The MAIN PART  ////////////////////////
////////////////////////////////////////////
        try {

           Extent ext = new Extent("lv.tsi.oodb.db.orders.Status");
           Status stat = new Status();

           while (ext.hasNext())  {
              stat = (Status)ext.next();
              if (stat.getStatusName()=="New") {
                break;
              }

           }

           User b = new User();
           ext = new Extent("lv.tsi.oodb.db.orders.User");

           while (ext.hasNext())  {
              b = (User)ext.next();
              if (b.getName()=="Arnold Z") {
                break;
              }

           }

           User supp = new User();
           ext = new Extent("lv.tsi.oodb.db.orders.User");

           while (ext.hasNext())  {
              supp = (User)ext.next();
              if (supp.getName()=="Robert M") {
                break;
              }

           }

           Delivery d = new Delivery();
           ext = new Extent("lv.tsi.oodb.db.orders.Delivery");

           while (ext.hasNext())  {
```

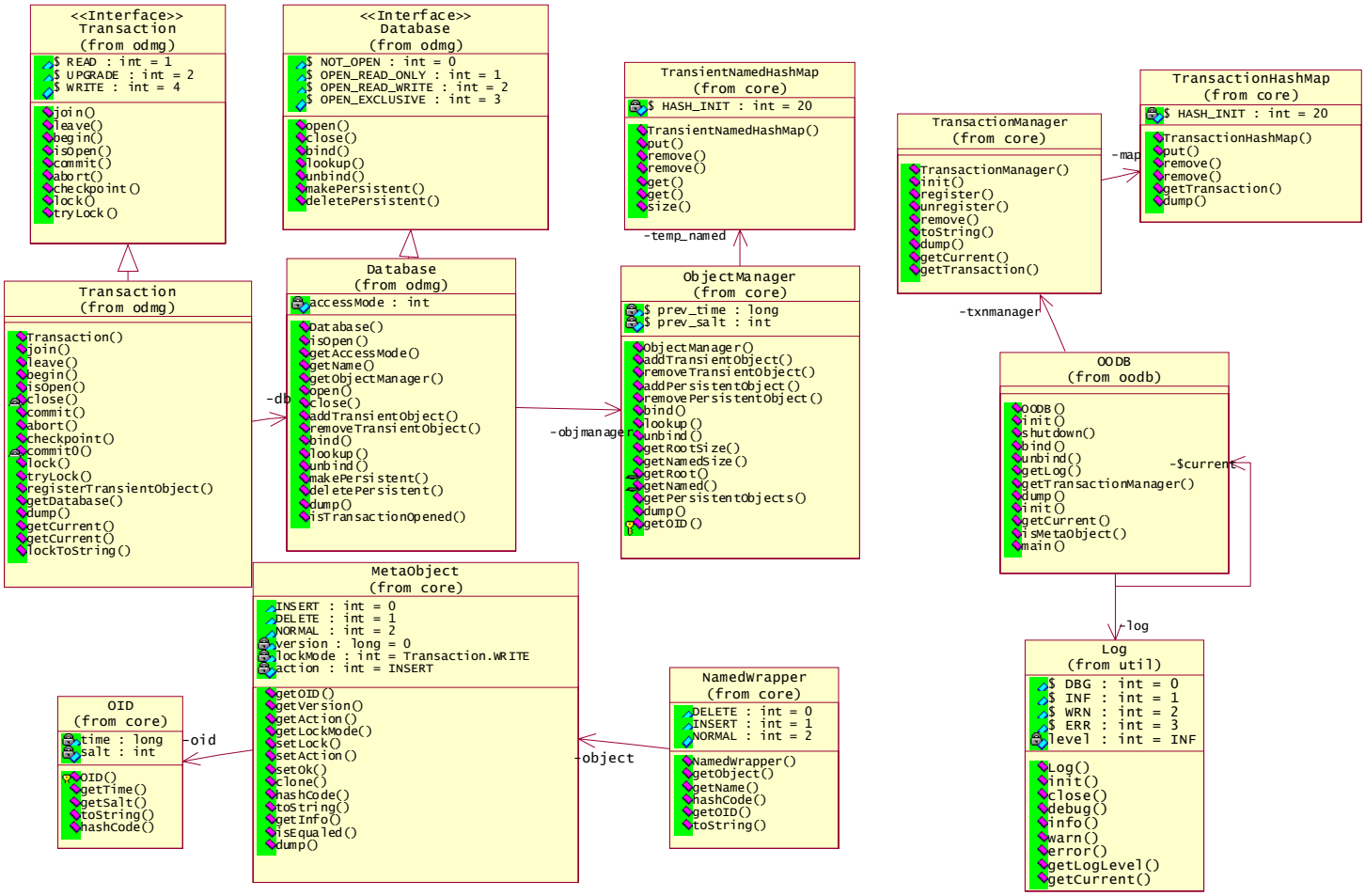
```
            d = (Delivery)ext.next();

            break;
          }

          Product p = new Product();
          ext = new Extent("lv.tsi.oodb.db.orders.Product");

          while (ext.hasNext())  {

            p = (Product)ext.next();
            if (p.getUpc()=="S45") {
             break;
            }
          }

          Order o = new Order();
          o.init( "6321-32423", "WEB" );

          o.addUser(b);
          o.addUser(supp);

           SubOrder s1 = new SubOrder();
           s1.init("6392-3819", "INVENTORY", "21", 0.5F,  "", "0x930232", "0x394222");
           s1.addUser(b);
           //s1.addUser(supp);

             Item i1 = new Item();
             i1.init(10, null, 300, 0.4F, "Telefon SIEMENS S45", 300);
             i1.linkToProduct(p);

           s1.addItem( i1 );
           s1.linkToStatus(stat);
           s1.linkToDelivery(d);

          o.addSubOrder( s1 );
          o.linkToStatus(stat);


          db.bind( o, "18Dec2000");

        } catch (Exception exc) {
            // in case something went wrong, we abort the transaction
            // this is especially important to release all locks
            // then we "rethrow" the exception
            txn.abort();
            throw exc;
        }
////////////////////////////////////////////////
////////////////////////////////////////////////
        txn.commit();
        System.out.println("Query time: " + (System.currentTimeMillis() - time)/1000 + " sec." );
        txn.dump();

        db.dump();


        // ------------------------------------------------
        // | Closing the database
        // ------------------------------------------------
        try {
                System.out.print( "Closing database..." );
                db.close();
```

```
            System.out.println( "ok." );
        } catch (ODMGException e) {
            System.out.println( "error.\nExited." );
            System.out.print( e.getMessage() );
        }

            }

}
```

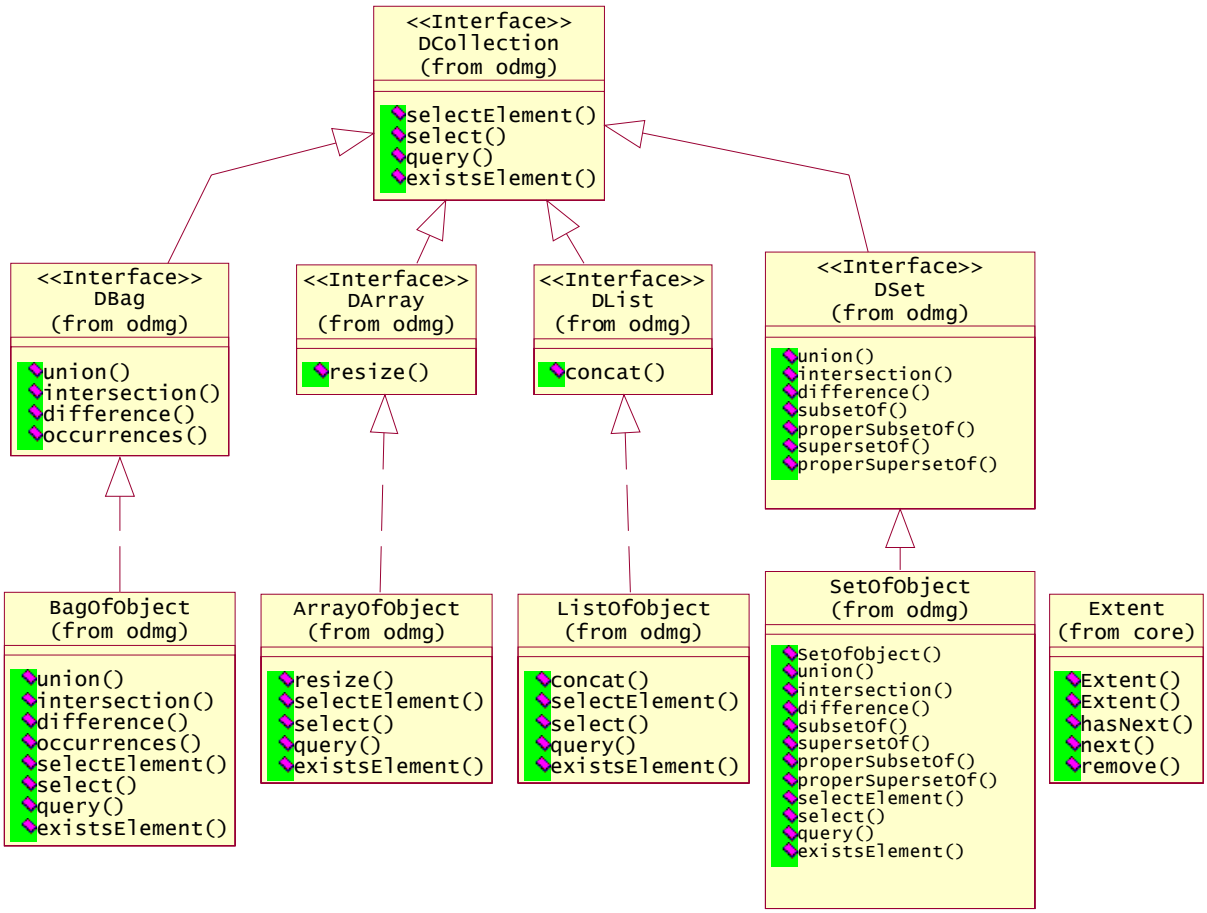
6. Используемая литература

1. Atkinson, M., Bancilhon, F., DeWitt, D., Dittrich, K., Maier, D., Zdonik, S.: *The Object-Oriented Database System Manifesto*. in Kim, W., Nicolas, J-.M., Nishio, S., eds., Proceedings of the First International Conference on Deductive and Object-Oriented Databases (DOOD), Elsevier Science Publishers, Amsterdam, 1989, 40-57.
2. M. Stonebraker, *Third-Generation Database Manifesto*, SIGMOD Record, Vol. 19, No. 3, Sept. 1990.
3. Cattel R.G.G.: *The Object Database Standard: ODMG*-93, Release 1.2, Morgan Kaufmann Publishers, Inc., 1994.
4. R. G. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez: "*Object Data Standard. ODMG 3.0*." January 2000 -- 300 pages – paper ISBN 1-55860-647-5 .
5. *POET Java™ Programmer's Guide*, OSS-JVPG-10MAY00. POET Software Corporation, San Mateo; POET Software GmbH, Hamburg, 2000.
6. Джеффри Д.Ульман, Дженнифер Уидом. *Введеление в системы баз данных*. Издательство "Лори", Москва, 2000. – 374 с.

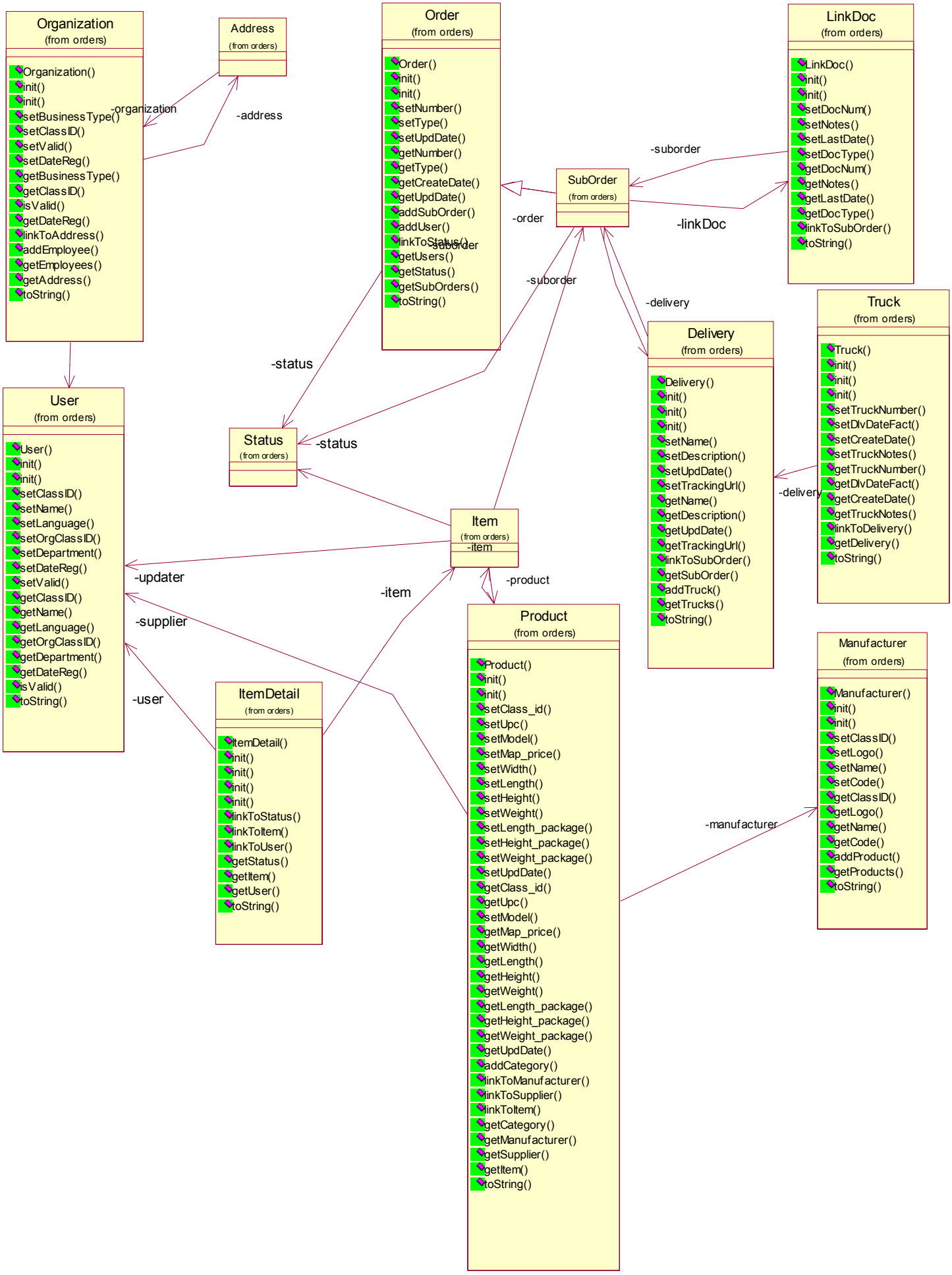
7. Приложение 1. Диаграмма классов ядра СУООБД



8. Приложение 2. Диаграмма классов типов данных ООБД



9. Приложение 3. Диаграмма классов ООБД Buyer-Supplier



10. Приложение 4. Стандарт ODMG 3.0

ODMG 3.0 разработан в январе 2000 года и является продожением ранних стандартов ODMG 1.2 (1993) и ODMG 2.0 (1997). Спецификация данного стандарта доступна в книжном варианте:

"*Object Data Standard. ODMG 3.0*." January 2000 -- 300 pages – paper ISBN 1-55860-647-5 . Edited by R. G. G. Cattell, Douglas K. Barry, Mark Berler, Jeff Eastman, David Jordan, Craig Russell, Olaf Schadow, Torsten Stanienda, and Fernando Velez. (~40$)

Помимо описательной теории этот стандарт определяет связывания (binding) с тремя языками программирования: C++, Smalltalk и Java. Связывание – это набор интерфейсов для взаимодействия с ООБД на высокоуровневых языках программирования.

Содержание стандарта