# Сравнительный анализ способов организации хранения информационных объектов с динамической структурой

Д.Г. Леонов

**Аннотация.** Рассматривается задача организации хранения информационных объектов в программно-вычислительных комплексах моделирования. Описываются проблемы, возникающие при использовании традиционного способа сериализации, а также решения, обеспечивающие отображение динамической объектной модели в реляционную СУБД.

**Ключевые слова:** моделирование, базы данных, объектно-реляционное отображение.

## Введение

Единая система газоснабжения (ЕСГ) России имеет иерархическую структуру, управление которой основано на тесном взаимодействии диспетчерских служб различных уровней. Автоматизированные системы диспетчерского управления являются сложными человеко-машинными (эргатическими) системами [1, 2]. В работе диспетчера важная роль отводится компьютерным комплексам, моделирующим объект управления.

Одним из общепринятых подходов, используемых при моделировании сложных систем, стало объектно-ориентированное представление предметной области, при котором реальный технологический объект моделируется информационным объектом, обладающим соответствующей структурой и поведением. По мере возрастания сложности объектной модели растет влияние выбора способа хранения объектной модели на эффективность функционирования системы в целом. Рассмотрим задачу обеспечения адекватного способа хранения объектной модели на примере программно-вычислительного комплекса (ПВК) моделирования газотранспортных систем «Веста», разработанного в РГУ нефти и газа имени И.М. Губкина.

## 1. Проблемы обеспечения устойчивости объектов с помощью сериализации

В терминах объектно-ориентированного подхода устойчивость — одна из основных характеристик информационного объекта, обеспечивающая возможность его существования во времени и пространстве независимо от процесса, породившего объект [3]. Типовым способом реализации устойчивости является сериализация, обеспечивающая сохранение объекта в виде байтовых последовательностей с их последующим восстановлением. Соответствующие возможности доступны во всех современных объектно-ориентированных средствах разработки либо на уровне стандартных реализаций, либо с помощью вспомогательных библиотек и модулей (Boost Serialization, MFC Serialization в C++, System.Runtime. Serialization и System.Xml.Serialization в .NET, pickle в Python, Storable в Perl и т.п.).

Данный подход, как правило, подразумевает полное считывание объектной модели в оперативную память процесса. Это обеспечивает максимальное быстродействие, но по мере роста числа объектов и задач, связанных

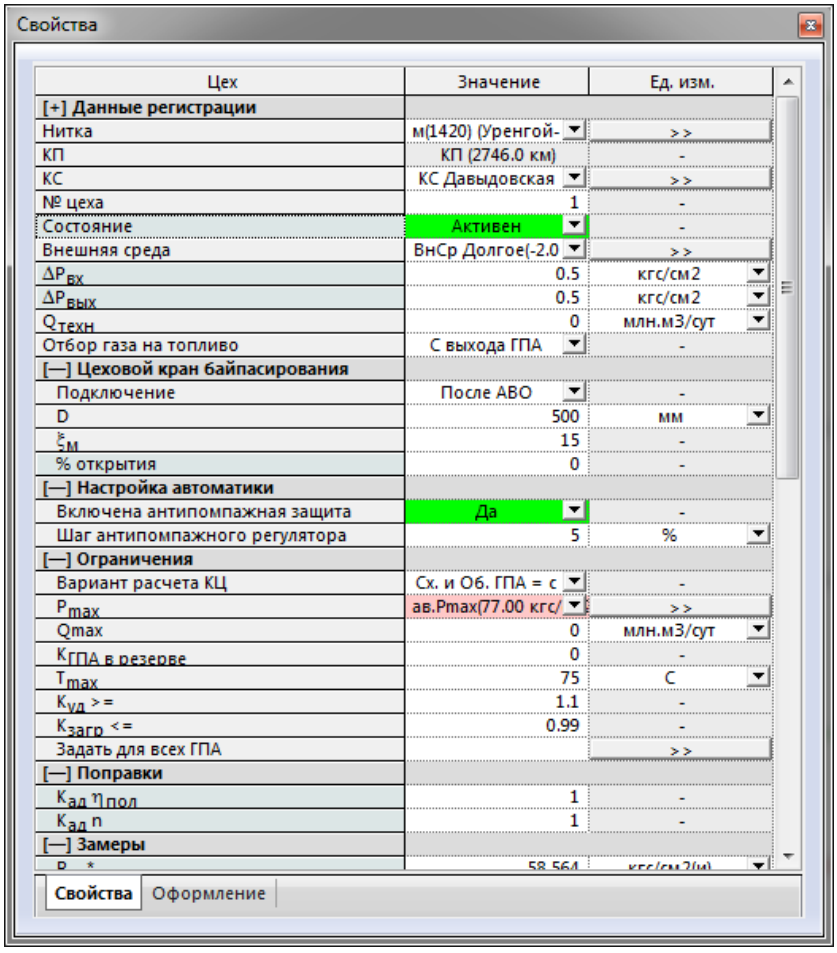


Рис. 1. Свойства информационного объекта «компрессорный цех»

с обработкой нестационарных процессов на длительных интервалах времени, такое решение становится неприемлемым из-за ограничений на доступные объемы оперативной памяти, характерные для все еще активно используемых на предприятиях 32-битных операционных систем.

В случае ПВК «Веста» средняя технологическая схема содержит порядка 20-30 тысяч объектов, количество свойств (включающих как исходные данные, так и результаты расчета), каждый из которых может меняться в диапазоне 10-20 (простые и вспомогательные объекты, непосредственно не участвующие в моделировании, такие как перемычки и объекты визуализации) до 50-60 (агрегаты, цеха, линейные краны и т.п.), как показано на Рис. 1.

Обработка объемов данных, необходимых для решения задач моделирования стационарных режимов транспорта газа, не вызывает особых сложностей. При переходе к задачам моделирования нестационарных режимов нагрузка на память несколько возрастает за счет того, что для 2-3 тысяч объектов необходимо сохранять динамику по 10-20 свойствам, но в случае ограничения диапазона моделируемого времени несколькими часами при шаге моделирования в одну минуту объем данных остается достаточным для размещения в оперативной памяти.

Общий объем памяти, необходимой для размещения лишь свойств, непосредственно принимающих участие в расчетах, без учета вспомогательных строковых свойств и накладных расходов, можно приблизительно оценить как:

$$M = N\,(S_i\,N_{pi} + S_d\,N_{pd}) + \\ +N_t\,N_{dyn}\,(S_i\,N_{dpi} + S_d\,N_{dpd}),$$

где N – общее количество объектов,

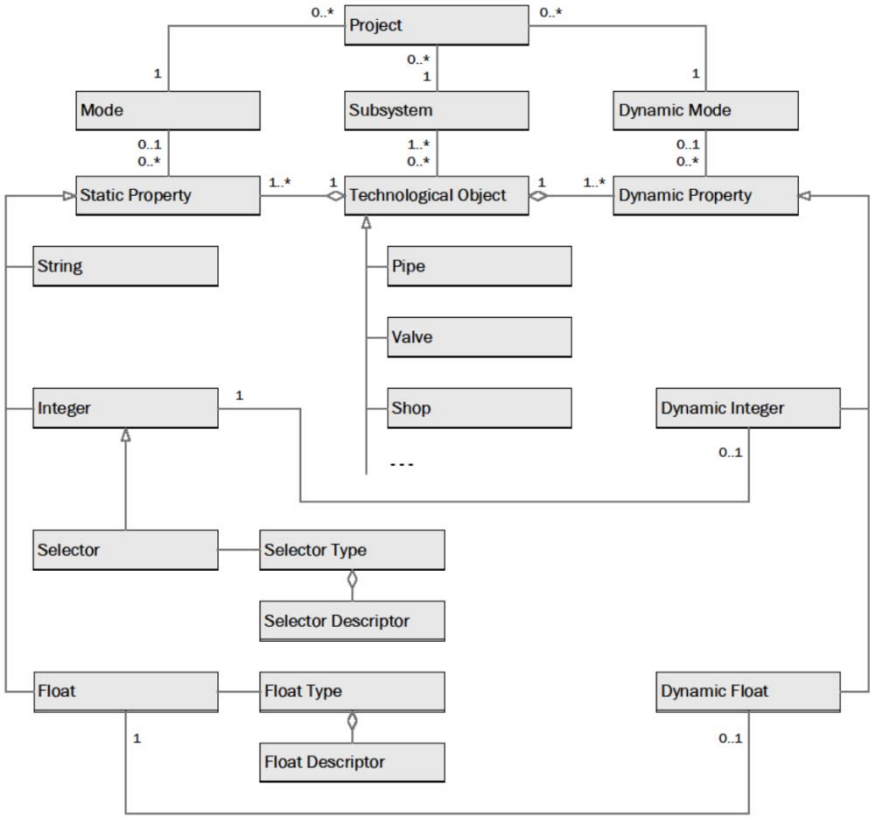


Рис. 2. Диаграмма классов

$N_{dyn}$ – количество объектов, содержащих динамическую информацию,

$S_i$ и $S_d$ – размер представления целочисленных (int, 4 байта) и вещественных (double, 8 байт) типов данных соответственно,

$N_{pi}$ и $N_{pd}$ – среднее число целочисленных и вещественных свойств соответственно,

$N_{dpi}$ и $N_{dpd}$ – среднее число динамических целочисленных и вещественных свойств соответственно,

$N_t$ – количество рассчитываемых временных слоев (60 в час при шаге в минуту).

Для технологической схемы с приведенными выше характеристиками можно получить оценку объема расчетной информации в 15-25 Mb на час расчетного времени.

Опыт эксплуатации ПВК, а также создание на его основе тренажерного комплекса потребовали расширения объектной модели с помощью следующих элементов [4]:

- *подсистемы*: позволяют выделять подмножества технологических схем и задавать принадлежность к ним отдельных объектов;
- *базовые режимы*: обеспечивают работу с различными наборами оперативных данных, не меняющихся в динамике и привязанных к одним и тем же множествам объектов (например, для перехода от зимнему к летнему режиму работы без повторного считывания схемы);
- *динамические режимы*: обеспечивают работу с различными наборами динамических данных (изменения свойств объектов и аварийные ситуации для различных учебно-тренировочных задач (УТЗ), действия пользователя и т.п.);
- *проекты:* обеспечивают связь между подсистемами, базовыми и динамическими режимами.

На Рис. 2 представлена упрощенная UML-диаграмма, описывающая взаимодействие между основными классами технологической схемы ПВК «Веста» с учетом описанных изменений. В данной объектной модели свойства информационных объектов являются не полями соответствующих классов C++, а самостоятельными объектами, список которых хранится в каждом

информационном объекте, т.е. сама структура информационных объектов является динамической. Это решение обеспечивает:

- возможность изменения структуры объектов без модификации исходного кода;
- возможность задания различных значений одного и того же свойства для разных режимов;
- возможность работы с технологическими схемами, сохраненными предыдущими версиями комплекса.

С учетом добавления в модель режимов и динамических режимов оценка (1) преобразуется в (2):

$$M = N_m N (S_i N_{pi} + S_d N_{pd}) + N_{dm} N_t N_{dyn} (S_i N_{dpi} + S_d N_{dpd}), \quad (2)$$

где $N_m$ – количество базовых режимов,

$N_{dm}$ – количество динамических режимов.

При подготовке УТЗ инструкторы могут создавать десятки подсистем, базовых и динамических режимов. Для 5-10 базовых и динамических режимов мы получаем уже оценку 75-550 Mb на час расчетного времени. К еще более существенному росту объемов данных приводит переход к находящемуся на стадии тестирования расчету с шагом в одну секунду, позволяющему более адекватно моделировать переходные процессы, возникающие при переключении кранов, запуске газоперекачивающих агрегатов и т.п. Полученные значения увеличиваются еще в 60 раз (4.5-34 Gb), что превышает как возможности 32-битных приложений, так и общий объем оперативной памяти, доступной на большинстве современных рабочих станций.

Поскольку в каждый момент времени используется лишь часть этих данных, загружать их целиком в оперативную память становится нецелесообразно. Но частичная сериализация значительно осложняет задачу обеспечения согласованности данных: например, удаление либо копирование объекта должно привести к удалению либо дублированию всех связанных с ним свойств вне зависимости от того, принадлежат ли они загруженному в данный момент режиму или нет.

Таким образом, возникает задача обеспечения хранения значительных объемов данных в сочетании с оперативным доступом к ним и возможностями манипулирования, характерными для СУБД. Переход с сериализации на использование сетевых клиент-серверных СУБД также позволяет упростить задачу организации взаимодействия в распределенных вычислительных комплексах и интеграции в единое информационное пространство предприятия [4, 5].

## 2. Задача реализации объектной модели средствами СУБД

Наиболее естественным способом переноса объектной модели в базу данных является использование объектно-ориентированных СУБД (ООСУБД), интерес к которым возрос вместе с ростом популярности объектно-ориентированного подхода. В данном случае объектная модель данных приложения и СУБД полностью совпадают, а все объекты строго типизированы. Обеспечивается встроенная поддержка наследования и отношений «многие ко многим».

Однако использование ООСУБД для решения описанной задачи сталкивается с двумя проблемами. Во-первых, несмотря на способность ООСУБД воспроизводить достаточно сложные объектные модели, они ориентированы либо на манипуляции объектами с помощью встроенных языков, либо на полное считывание объектов в клиентские приложения. В случае использования встроенных языков возникает необходимость существенной модификации программного кода с жесткой привязкой к выбранной ООСУБД. Полное считывание объектов с точки зрения обсуждаемой проблемы принципиально не отличается от традиционной сериализации.

Во-вторых, несмотря на многочисленные исследования и десятки коммерческих реализаций, стандартизация в этой области значительно уступает стандартизации в области традиционных реляционных СУБД [6]. Объектно-ориентированные расширения, реализованные рядом производителей популярных реляционных СУБД, также зачастую оказываются несовместимыми между собой. С учетом того, что комплекс моделирования, как правило, встраивается в уже существующую инфраструктуру, установка у пользователей подобных нестандартных решений не всегда является возможной.

Традиционные реляционные СУБД не имеют подобных проблем с совместимостью на уровне базовых возможностей языка SQL, поэтому популярным подходом стало представление объектов в реляционных СУБД с помощью систем класса ORM (Object-relational mapping, объектно-реляционное отображение): ADO.NET Entity Framework, LINQ to SQL, ActiveJDBC, ORMLite и т.п.

Выбор существующих ORM-библиотек для C++ значительно уже, чем для .NET или Java. Были рассмотрены четыре решения: LiteSQL (http://litesql.sf.net/), ODB (http://codesynthesis.com/products/odb/), QxOrm (http://qxorm.com/), Wt::Dbo (http://webtoolkit.eu/). Во всех перечисленных библиотеках осуществляется связь непосредственно между членами классов C++ и полями таблиц базы данных. Например, при использовании LiteSQL структура базы задается в отдельном xml-файле, при использовании ODB размечается непосредственно в коде приложения. Далее запускается специальный препроцессор, генерирующий вспомогательный код, который должен быть скомпилирован вместе с основным кодом приложения. Поэтому эти библиотеки непригодны для непосредственного отображения в БД динамической структуры информационных объектов.

Наиболее функциональной из рассмотренных является библиотека ODB, и ее можно рекомендовать для использования во всех ситуациях, когда при изменении структуры базы есть возможность перекомпилировать исходный код. QxOrm, Wt::Dbo зависят от более крупных библиотек, использование которых не всегда оправдано, в LiteSQL и Wt::Dbo отсутствует поддержка как Microsoft SQL, так и ODBC.

Для обеспечения интеграции с используемыми на предприятиях СУБД в ПВК «Веста» был реализован модуль, абстрагирующий взаимодействие с реляционной базой данных с помощью вспомогательного адаптера. Основным механизмом взаимодействия является ODBC, также реализованы адаптеры для SQLite, DAO, Oracle OCCI.

Было проведено сравнение двух подходов к отображению динамической структуры в реляционную базу: с использованием строго реляционной и «развернутой» моделей.

В первом случае модель данных полностью соответствует приведенной ранее диаграмме классов, с заменой отношений «агрегирование» на отношения «один ко многим» и добавлением таблиц с метаинформацией о динамической структуре (Рис. 3). Все свойства одного типа сохраняются в одной таблице, состав свойств отдельного объекта определяется с помощью связей с таблицами, хранящими метаинформацию.

Подобная организация БД обеспечивает максимальную универсальность и гибкость, позволяющую учитывать изменения в структуре объектов без изменения структуры таблиц. Но она приводит к большим затратам как при выборке объектов за счет интенсивного использования оператора join, так и при вставке объектов из-за значительного количества индексируемых записей.

Для технологических схем с приведенными выше характеристиками данная структура БД приводит к формированию таблиц статических свойств, содержащих $2\text{-}6\cdot10^6$ записей. В таблицах динамических свойств каждый час моделируемого времени требует $6\text{-}55\cdot10^6$ записей для расчетов с минутным шагом и $0.35\text{-}3\cdot10^9$ записей при секундном шаге. Тесты с использованием Mictosoft SQL и SQLite продемонстрировали существенное снижение производительности уже на объемах, имитирующих схему с 10 тысячами объектов.

Таблицы динамических свойств могут быть сокращены за счет пропуска записей, соответствующих значениям, не изменившимся по сравнению с предыдущим временным слоем. Но это значительно усложняет обработку при выборке произвольного временного слоя и не дает существенного выигрыша при моделировании переходных процессов (например, при переходе с одного технологического режима на другой).

Более эффективным способом оптимизации является «разворачивание» строго реляционной модели с преобразованием единой таблицы свойств в отдельные таблицы, соответствующие типам информационных объектов (Рис. 4). В этом случае каждому типу объектов соответствуют три таблицы свойств (динамические, привязанные и не привязанные к режиму) и одна таблица с метаинформацией.

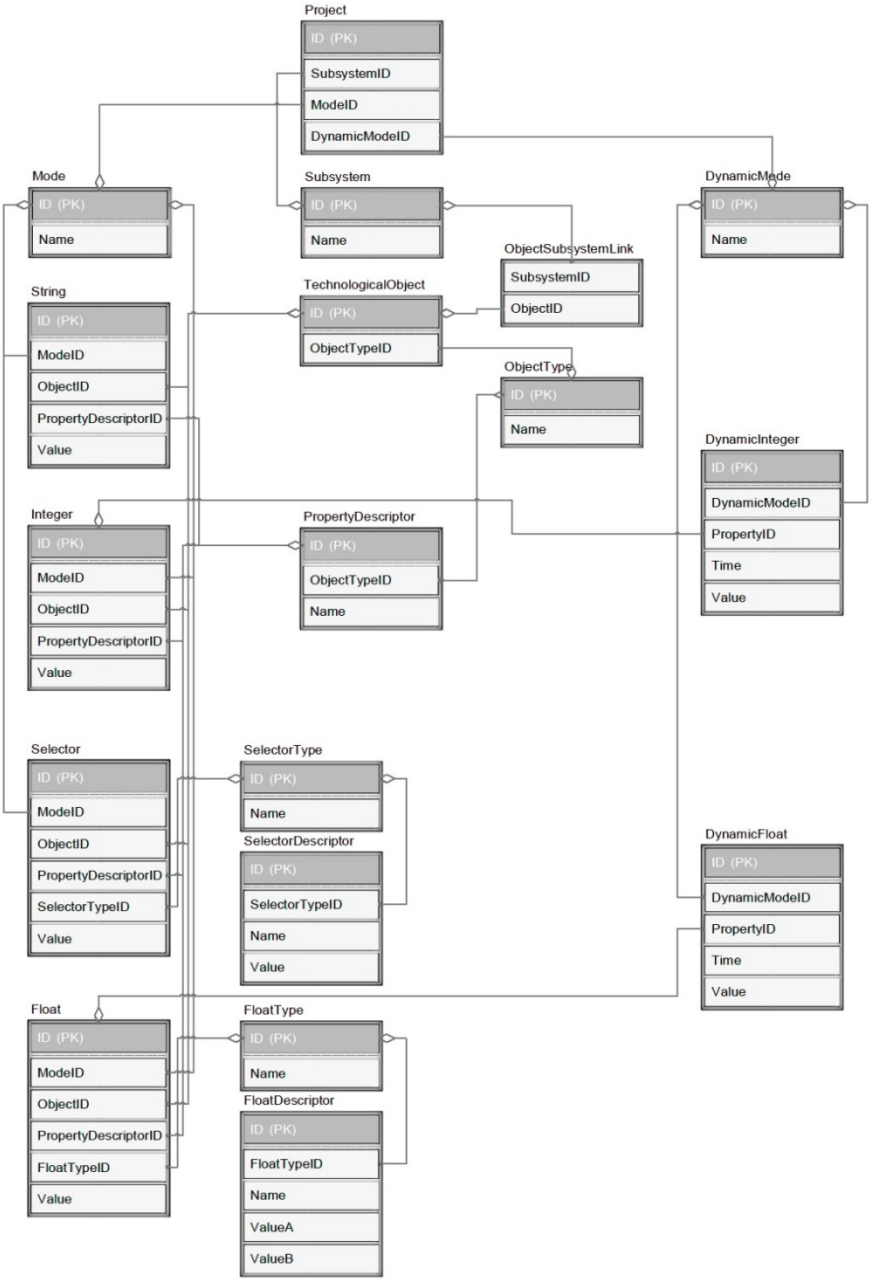


Рис. 3. Реляционная модель данных в нотации IDEF1X

Данная реализация подразумевает обязательную автоматическую проверку актуальности структуры БД при каждом подключении и в случае необходимости приведение ее в соответствие с текущей конфигурацией.

Итак, после рассмотрения возможных подходов к обеспечению устойчивости объектной модели технологических схем в качестве основного было выбрано решение, основанное на ее объектно-реляционном отображении в «развернутую» реляционную модель. Данное решение обеспечивает приемлемые показатели производительности при сохранении достаточной гибкости и обратной совместимости с ранее созданными технологическими схемами.

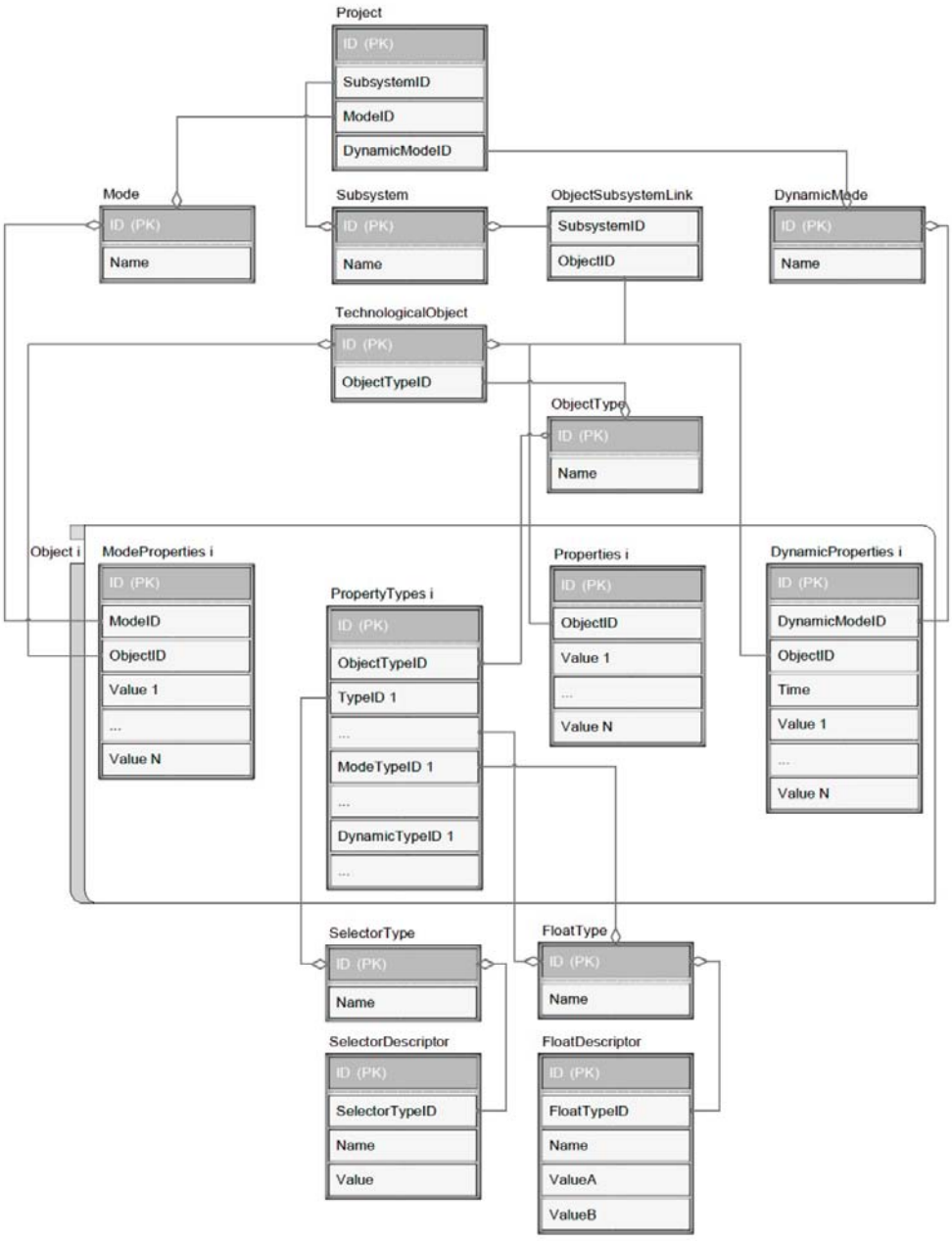


Рис. 4. «Развернутая» модель данных в нотации IDEF1X

## Литература


1. Григорьев Л.И., Пирогов А.В. Синергетический анализ реализации жизненного цикла проектирование-производство-эксплуатация автоматизированных систем диспетчерского управления в нефтегазовом комплексе. М.: Автоматизация, телемеханизация и связь в нефтяной промышленности. №3, 2012 г.
2. L. Grigoriev, A. Kostogryzov, A. Tupysev. Automated dispatch control; problems and details of modeling. Preprints of the 2013 IFAC Conference on Manufacturing Modelling, Management, and Control, Saint Petersburg State University and Saint Petersburg National Research University of Information Technologies, Mechanics, and Optics, Saint Petersburg, Russia, June 19-21, 2013.
3. Буч Г. Объектно-ориентированный анализ и проектирование с примерами приложений. М.: Вильямс, 2008 г.
4. Васильев А.В., Леонов Д.Г., Сарданашвили С.А., Швечков В.А. Интеграция расчетных комплексов моделирования режимов систем газоснабжения в единое информационное пространство АСДУ ОАО «ГАЗПРОМ». М.: Газовая промышленность. № 12, 2012 г.

5. Леонов Д.Г., Швечков В.А. Организация хранения данных в распределенном вычислительном комплексе при решении задач диспетчерского управления режимами ГТС. М.: Автоматизация, телемеханизация и связь в нефтяной промышленности. № 9, 2005 г.

6. К.Дж.Дейт, Хью Дарвен. Основы будущих систем баз данных. Третий манифест. М.:Янус- К, 2004 г.

**Леонов Дмитрий Геннадьевич**. Доцент кафедры АСУ РГУ нефти и газа им. И.М. Губкина. Окончил Государственную академию нефти и газа имени И.М. Губкина в 1992 году. Кандидат технических наук, доцент. Автор 35 печатных работ. Область научных интересов: распределенные вычислительные системы, объектно-ориентированный подход, моделирование газотранспортных систем. E-mail: dl@bugtraq.ru