Универсальные формы заполнения и представления научных электронных документов.

Косарик Валерий Валентинович
valera@a170.ipi.ac.ru, valery_lek@mail.ru.
ИПИ РАН

В статье рассматривается создание универсальных структур заполнения научных электронных документов, на основе индуктивного метода выявления свойств полей, форм, комплектов форм, и представление данных на основе дедуктивного метода создания набора данных, комплекта представлений, пригодных как для табличного, так и для графического вида.

1. Начальное состояние рассматриваемой темы.

Многие задачи управления данными не могут решаться независимо от формата представления данных, а также без привязки к той или иной программной среде, в которой работает разработчик и соответственно пользователь системы управления данными. Под управлением данными подразумеваются задачи:

- заполнение новыми данными;
- обновление данных;
- представление в удобном для пользователя виде, будь то таблицы, строки, графики, и другие объекты.
- удаление данных.

Задача представляющая истинный научный интерес заключается в разработке системы управления данными, которая была бы независима от формата данных и от среды программирования, ведь большинство используемых на данный момент времени форматов считаются «закрытыми» и поддерживаются только разработчиками.

Если надо сделать систему, которая будет работать на протяжении многих десятков лет, то не стоит опираться на фирмы-изготовители программных продуктов, так как существует тенденция, по которой фирма может быть «съедена» более крупным конкурентом либо по какой-нибудь другой причине вообще «пропадет».

Рассмотрим историю развития форматов хранилищ данных.

o Первые применения первых компьютеров.

Большинство работ по разработке компьютеров производилось по заказу военных и на их деньги. Естественно, что и в числе задач, решаемых на первых

компьютерах, превалировали расчетные задачи, а именно: для наводки дальнобойных орудий, траекторий ракет, моделирования цепных реакций ядерного распада, аэродинамических характеристик и других аналогичных. Все эти задачи базировались на таблицах.

Накладывали свои ограничения на характер решаемых задач и архитектурные особенности компьютеров тех лет: если ламповые схемы давали относительно высокое быстродействие (десятки-сотни тысяч операций в секунду), то запоминающие устройства имели крайне невысокую емкость в единицы-десятки килослов и ограниченное быстродействие. Делались они либо на ртутных линиях задержки (мизерная емкость), либо на электронно-лучевых трубках (более высокая емкость при высокой цене – порядка 1.000 долларов – и малой надежности – срок службы около месяца, что чрезвычайно удорожало использование компьютеров с большим объемом оперативной памяти).

Характеристики периферийных устройств также не поражали воображение. Весьма популярны были низкоскоростные перфокарточные и перфоленточные устройства ввода/вывода. Имелись также накопители на магнитной ленте и магнитных барабанах. Все эти носители информации, исключая последний вариант, делали возможной последовательную обработку данных, но были непригодны для выборки их в произвольном порядке. Магнитные же барабаны имели слишком малую емкость, хотя и обеспечивали произвольный доступ к данным.

- Появление коммерческих компьютеров.

Важнейшим событием, повлиявшим на дальнейшее развитие компьютеров, явилось изобретение оперативной памяти на ферритовых сердечниках. Такая память была достаточно компактна и надежна при невысокой стоимости. Довольно долго она использовалась в компьютерах, пока ее не вытеснила более практичная полупроводниковая память.

Эта и ряд других находок позволили существенно снизить себестоимость компьютеров и поставить их производство на серийную основу. Компьютеры начали выходить из стен университетских и военных лабораторий. Соответственно постепенно менялся круг решаемых ими задач.

Примерно в это же время появляются магнитные диски, в которых информация записывалась концентрическими кольцами — дорожками. Магнитные головки могли произвольно позиционироваться на любую дорожку специальным приводом, причем за сравнительно небольшое время (десятки миллисекунд). При этом емкость дисков достаточно быстро выросла с единиц до сотен мегабайт.

Параллельно шло развитие операционных систем. От простейших загрузчиков и редакторов связей они выросли до сложных многозадачных многопользовательских систем, от пакетных систем – до диалоговых.

Все это создало предпосылки для применения компьютеров в задачах управления и информационных системах, которые не сводились только к вычислениям.

- Организация данных на внешних носителях.

Нечисловая обработка данных потребовала уделить больше внимания структурированию хранимой информации. Эволюция структурирования данных выглядела примерно следующим образом.

- Плоские файлы.

Простейшим способом хранения информации являются плоские файлы. Практически любая из ныне используемых операционных систем (ОС) поддерживает плоские файлы, за исключением некоторых специализированных для встраиваемых контроллеров.

Плоский файл — это именованный набор данных на внешнем носителе. Сама ОС никакой структурой плоский файл не наделяет и трактует его просто как набор байт. Задача разделения последовательности байт на записи и выделения полей в них ложится целиком на прикладную программу.

Основные операции доступа к плоским файлам – открытие на чтение/запись, закрытие, позиционирование на начало файла/конец файла/заданный байт, чтение/запись заданного количества байт с текущей позиции.

Операционные системы , например, семейства UNIX или MS Windows, предоставляют намного более развитые средства для доступа к файлам. Тем не менее, все они имеют общие недостатки:

Для ОС плоский файл – всего лишь последовательность байт. Поэтому все предположения о структуре записей файла делает прикладная программа. Ошибка в коде или ошибочное обращение не к тому файлу приведет к непредсказуемым последствиям.

Как следствие – невозможность заблокировать на чтение/модификацию отдельную запись файла средствами ОС, так, как само понятие записи как таковое отсутствует.

Для поиска нужной записи по условию на значение полей придется считывать все записи последовательно с начала файла до тех пор, пока поиск не увенчается успехом, либо наступит конец файла. Поэтому о произвольной выборке данных в этом случае можно говорить с большой натяжкой.

Эти недостатки попытались преодолеть поддержкой индексно-последовательных файлов.

- Индексно-последовательные файлы (И-П файл)

При создании И-П файла создается так называемый индекс, который служит для быстрого доступа к записям по значению ключевого поля. Следует обратить внимание на то, что делит файл на записи и поддерживает индекс сама ОС, а не исполняющая система, скомпонованная с прикладной программой

Как следует из самого названия, с И-П файлами можно работать двояко. Во-первых, можно считывать записи последовательно, одну за другой. Во-вторых, можно сразу находить нужную запись по значению ключевого поля, при этом ОС использует индекс для ускорения поиска.

И-П файлы – большой шаг вперед по сравнению с плоскими файлами. Тем не менее, и они не лишены недостатков.

В одном файле могут храниться только сущности одного вида. Создание сложной модели данных приведет к нагромождению большого количества ничем не связанных между собой файлов.

В операционной системе отсутствуют средства для описания взаимосвязей между сущностями, хранящимися в различных файлах. Об этих взаимосвязях «знает» только прикладная программа, обрабатывающая данные.

- Системы управления базами данных (СУБД).

Недостатки, присущие традиционным файлам, сдерживали дальнейшее развитие информационных систем. Возникла потребность в инструментальных средствах, более адекватных решаемым задачам. Такими средствами явились СУБД.

В создании СУБД участвовало много фирм, известных и не очень. Каждый производитель имел собственную точку зрения на то, каким должен быть идеальный продукт. В результате возникло множество СУБД, ничего общего не имевших друг с другом. По мере использования СУБД одни идеи получали развитие, заимствовались , другие, наоборот, отмирали.

Довольно быстро оказалось, что, несмотря на внешние различия между системами, подавляющее большинство из них можно отнести к трем видам: иерархическим, сетевым и реляционным СУБД.

* * * *

В основе иерархических СУБД лежит довольно простая модель данных, которую можно представить в виде дерева — ациклического ориентированного графа особого вида.

Дерево состоит из вершин, каждая из которых, кроме одной, имеет единственную родительскую вершину и несколько (в том числе ни одной) дочерних.

Вершина, не имеющая родительской, называется корнем дерева. Вершины, не имеющие дочерних, называются листьями. Остальные вершины являются ветвями.

Иерархические базы данных наиболее пригодны для моделирования структур, по своей природе являющихся иерархическими. В качестве примеров можно привести воинские подразделения или сложные механизмы, состоящие из более простых узлов, которые в свою очередь тоже можно подвергнуть декомпозиции.

Вместе с тем существует значительное количество структур, не сводящихся к простой иерархии. Например, всем известное генеалогическое дерево, которое на самом деле не является деревом в строгом смысле, поскольку у людей по два родителя. О более сложных структурах и говорить не приходится.

Иерархические СУБД быстро прошли пик популярности, которая обусловливалась их простотой в использовании и ранним появлением на рынке, когда основные конкуренты еще не дозрели для коммерческого использования. Многочисленные недостатки сделали их неконкурентоспособными, и в настоящее время иерархическая модель представляет только исторический интерес.

* * * *

Сетевые СУБД, подобно иерархическим, также можно представить себе в виде ориентированного графа. Но в этом случае граф может содержать циклы, то есть вершина может иметь несколько родительских.

Такая структура значительно выразительнее и пригодна для моделирования гораздо более широкого класса задач. В этой модели вершины представляют собой сущности, а соединяющие их ребра – отношения между ними.

Сетевые СУБД имели гораздо больший успех и долго господствовали на рынке. В немалой степени их успеху способствовала энергичная деятельность Data Base Task Group (DBTG) Комитета по языкам программирования Conference on Data Systems Languages (CODASYL). Эта организация тщательно проработала спецификации сетевой модели и ее архитектуру, что позволило создать ряд успешных коммерческих продуктов, не последнее место среди которых занимал некогда весьма популярный COBOL.

70-е годы XX века фактически стали эпохой расцвета сетевой модели. Сетевые СУБД весьма прочно укрепились на рынке, и реляционной модели пришлось с боем завоевывать свое место под солнцем. В истории информатики навечно останется «Великий Спор»[1], который на самом деле явился решающим сражением сетевой и реляционной моделей. В рядах сторонников сетевой архитектуры был сам великий Чарльз Бахман, и только гений Эдгара Кодда позволил реляционной модели одержать победу.

* * * *

Реляционные СУБД являются в настоящее время самыми распространенными. Их реализации существуют на всех мало-мальски пригодных для этого платформах (от персональных компьютеров до мэйнфреймов), для всех операционных систем и для всех применений – от простейших продуктов, предназначенных для ведения картотек

[1] Анализ вклада Кодда в Великий Спор. http://www.citforum.ru/database/articles/codd_1.shtml

индивидуального пользования, до сложнейших распределенных многопользовательских систем.

Несмотря на такое пестрое разнообразие, все СУБД имеют общую основу – реляционную модель данных, разработанную Коддом в 70-х годах XX столетия. С виду эта модель довольно проста: база данных выглядит как простой набор взаимосвязанных таблиц. Но за внешней простотой кроется мощный и вместе с тем изящный математический аппарат реляционной алгебры, которая в свою очередь базируется на целом ряде математических дисциплин, среди которых – логика, исчисление предикатов, теория множеств…

Немалую роль в успехе реляционных СУБД играет также язык SQL???, разработанный специально для запросов к реляционным БД. Это достаточно простой и в то же время выразительный язык, при помощи которого можно выполнять достаточно изощренные запросы к базе.

Разумеется, предшествующие СУБД также имели языки описания данных (ЯОД) и языки манипулирования данными (ЯМД). SQL объединил в себе обе эти функции. Но самой привлекательной его особенностью для пользователей-непрофессионалов в программировании, является то, что можно строить запросы на основе непроцедурного подмножества SQL. Это означает, что в формулировке запроса указывается, что должно содержаться в результате, а не как его получить. Имеются и процедурные элементы языка, например, операторы организации ветвления и циклов, но их применения зачастую удается избежать. При работе же с сетевыми БД программист был вынужден использовать навигационные процедуры, отвлекаясь при этом от решения самой задачи.

Любой разработчик хранилища данных сталкивается с проблемой построения Базы Данных (БД) для хранения информации, а также создания форм для доступа и пополнения данными своего хранилища.

2. Эра Интернет.

С появлением сети Интернет, появились новые возможности представления данных.

Новые реализации реляционных СУБД (MySQL, PostgreSQL, и т.д.).

А также появились новые языки описания содержания документов:

HTML (HyperText Markup Language) - гипертекстовый язык разметки документов (является упрощенной веткой языка SGML), принятый в World Wide Web для создания и публикации Веб-страниц. HTML предоставляет авторам средства для:

- включения в Веб-документы заголовков, текста, таблиц, списков, фотографий и т.п.;
- перехода к другим Веб-страницам посредством щелчка кнопки мыши по гипертекстовой ссылке;

- создания и заполнения форм для транзакций с удаленными службами, например, для поиска информации, бронирования билетов, оформления заказов на товары и др.

- непосредственного включения в Веб-документы видеоклипов, звука и других внешних объектов.

Но при всех положительных сторонах, имеются отрицательные:

HTML не выражает смысла документов.

Язык HTML был создан для описания *структуры* документов (название, заголовки, списки, абзацы и т. п.) и, в некоторой степени, правил их *отображения* (полужирный шрифт, курсивный шрифт и т. п.). Он ни в коей мере не предназначен для описания *смысла* написанных на нем документов, а во многих случаях именно данные составляют существо документа, будь-то биржевая сводка или научная публикация. Поэтому появилась необходимость в языке описания данных, причем данных, организованных в иерархические структуры.

HTML громоздок и негибок.

За последние годы HTML превратился в нагромождение тегов, которые часто дублируют друг друга и отнюдь не вносят ясности в текст документа. Если добавить сюда еще и нестандартные расширения HTML, которыми грешат все разработчики обозревателей, то создание мало-мальски сложных HTML-документов становится серьезной задачей. С другой стороны, раз и навсегда зафиксированный набор тегов часто оказывается недостаточно гибким для выражения нужного нам содержания.

Концепция Веб-обозревателя слишком ограничена.

С появлением Java-аплетов, сценарных языков и элементов ActiveX Веб-обозреватели перестали быть простыми "отображателями" HTML-документов; сегодня скорее они выглядят как программы, запускающие конкретные приложения. Тем не менее, сама концепция обозревателя накладывает излишние ограничения на пользователя; во многих случаях нужны *Веб-ориентированные приложения*, т. е. программы, способные читать специализированную информацию с Веб-узлов и выдавать ее в привычном виде, например, в виде электронных таблиц.

Поиск документов возвращает слишком много ссылок.

Все мы постоянно пользуемся поисковыми системами и постоянно клянем их за неудобство работы. Допустим, что мне нужны все тексты книг Филипа Дика, имеющиеся в Сети. Попытка поиска по имени автора приведет к тому, что я получу список всех ссылок с этим именем, включая воспоминания о Дике, рецензии на его книги и т. д. Намного удобнее было бы воспользоваться специальным тегом <AUTHOR>, чтобы указать, что именно я ищу.

Невозможно найти взаимосвязанные ресурсы.

Допустим теперь, что я все же нашел несколько рассказов Дика, которые явно составляют единый сборник. Хорошо, если они содержат ссылку на оглавление, но часто это не так. Поэтому необходим способ указания того, что данная группа страниц составляет единый ресурс и должна обрабатываться

соответственно. Для этого необходима стандартизованная и развитая система *метаописателей* Веб-страниц.

Поэтому рассмотрим другие возможности:

XML (eXtensible Markup Language) – расширяемый язык разметки данных, открытый формат представления именно структуры данных, при работе необходимо учитывать.

XML — это упрощенный диалект языка SGML, предназначенный для описания иерархических структур данных в World Wide Web. Он разрабатывается рабочей группой W3C с 1996 г.; в настоящее время принятой рекомендацией является вторая редакция языка XML 1.0 (октябрь 2000 г.), на которую и ориентируется дальнейшее изложение.

XML, несомненно, входит в обойму наиболее перспективных технологий WWW, чем объясняется интерес, который уделяется ему и корпорациями-разработчиками, и пользователями. Прежде чем перейти к его описанию, представляется уместным обсудить причины его появления и последующего бурного развития. Попытаемся для этого взглянуть на те проблемы WWW, которые должны быть решены средствами нового поколения Веб-технологий.

XML — это попытка решить перечисленные проблемы путем создания простого языка разметки, описывающего произвольные структурированные данные. Точнее говоря, это метаязык, на котором пишутся специализированные языки, описывающие данные определенной структуры. Такие языки называются *XML-словарями*. В отличие от HTML, XML не содержит никаких указаний на то, как описанные в XML-документе данные должны отображаться. Способ отображения данных для различных устройств задается языком описания стилей XSL, который играет для XML примерно ту же роль, что CSS для HTML. Другое принципиальное его отличие от HTML состоит в том, что XML может содержать любые теги, которые сочтут нужным использовать создатели XML-словаря. Приведем список лишь нескольких специализированных языков на базе XML, которые сегодня находятся в разных стадиях разработки рабочими группами W3C:

- MathML — язык математических формул;
- SMIL — язык интеграции и синхронизации мультимедийных средств;
- SVG — язык двумерной векторной графики;
- RDF — язык метаописаний ресурсов;
- XHTML — переформулировка HTML в терминах XML.

Процесс обработки XML-документа состоит в следующем. Его текст анализируется специальной программой, которая называется *XML-процессором*. XML-процессор ничего не знает о семантике данных в документе; он только производит синтаксический разбор (parsing) текста документа и проверяет его правильность с точки зрения правил XML. Если документ *правильно оформлен* (well-formed), то результаты разбора текста передаются XML-процессором прикладной программе, которая выполняет их содержательную обработку; если же документ оформлен неверно, то есть

содержит синтаксические ошибки, то XML-процессор должен сообщить о них пользователю.

Но при рассмотрении работы с XML необходимо помнить следующее:

1. Является семейством, под которым подразумевается множество функциональных разветвлений (Рисунок 1. Семейство XML.);

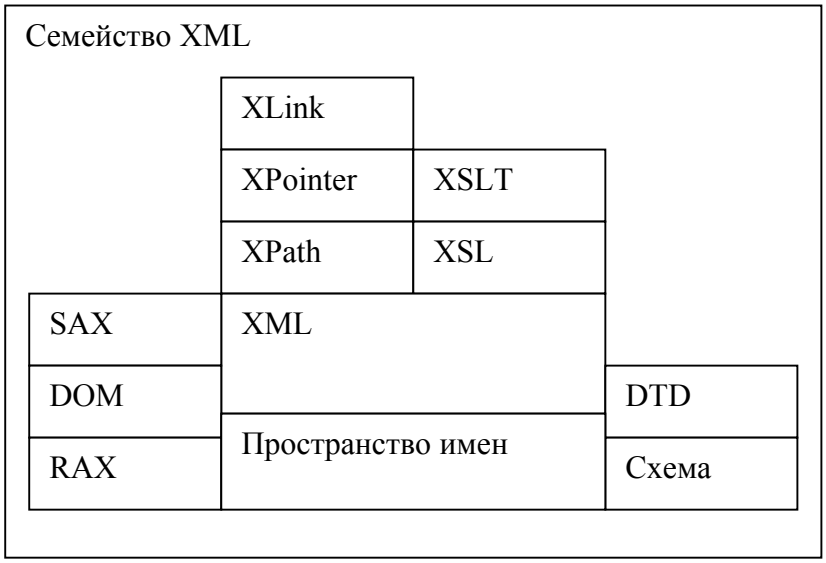

Рисунок 1. Семейство XML.

2. XML – является языком, самоописывающихся данных;
3. Легок для переноса между платформами;
4. Возможностью ввода специальных символов (математических и химических) за счет поддержки семейств.

Но при переходе к работе с этим семейством необходимо помнить парадоксы, описанные в статье Дона Петерсона, «Является ли XML панацеей?»[2].

Таблица 1. Сравнительная таблица xml и Баз Данных.

XML	База данных
XML хорошо подходит для описания как простых, так и сложных форматов данных. Особенно пригоден он для описания данных, использующих *динамические/сложные/вложенные* структуры, такие как docbook.	Базы данных хорошо подходят для хранения и извлечения «линейных» структур данных, которые можно представить в табличном формате.
Анализ/использование данных XML требуют значительных ресурсов. Простота/гибкость формата снижает производительность.	Базы данных позволяют значительно быстрее писать и извлекать данные. Структурированная природа данных улучшает производительность ценой потери гибкости.
XML очень легко передавать.	Перемещать базы данных труднее.
Человек может писать и читать XML, хотя лучше это делать при помощи редактора XML.	Мало кто способен вручную читать и писать файлы баз данных.

Исходя из Таблицы 1 выбираем, как инструмент для работы XML и его семейства.

[2] Дон Петерсон, «Является ли XML панацеей?», http://newsletter.narod.ru/sql_pages/sql_jul_2005.htm

3.Инвариантное индуктивно-дедуктивное решение задачи

Хорошо, если заранее знать все входные и выходные данные, но что делать, если разработчик толком не знает, какую структуру, он должен получить в законченном варианте, или, допустим, приходится менять структуру и наполнение базы данных. Задача заключается в том, чтобы создать независимую, от выбора пользователя той или иной платформы операционной системы и программирования, систему ввода научных данных.

Помимо этого, есть еще одна немаловажная деталь: на сегодняшний день нет рабочего стандарта для представления электронного научного документа, поэтому и возникла необходимость разработки подобной идеи.

Инвариантное индуктивно-дедуктивное решение задачи заключается в достижении глобальной цели возможности перехода «от меньшего - к большему» и « от большего – к меньшему».

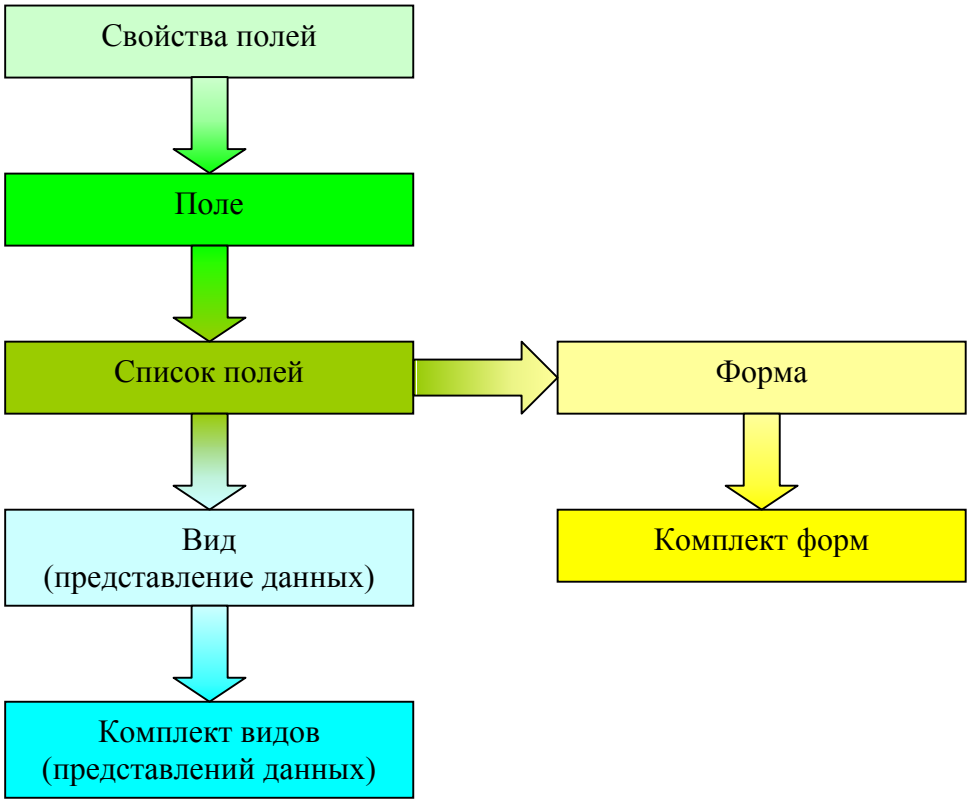

Рисунок 2. Общая концепция индуктивного метода перехода «от меньшего к большему».

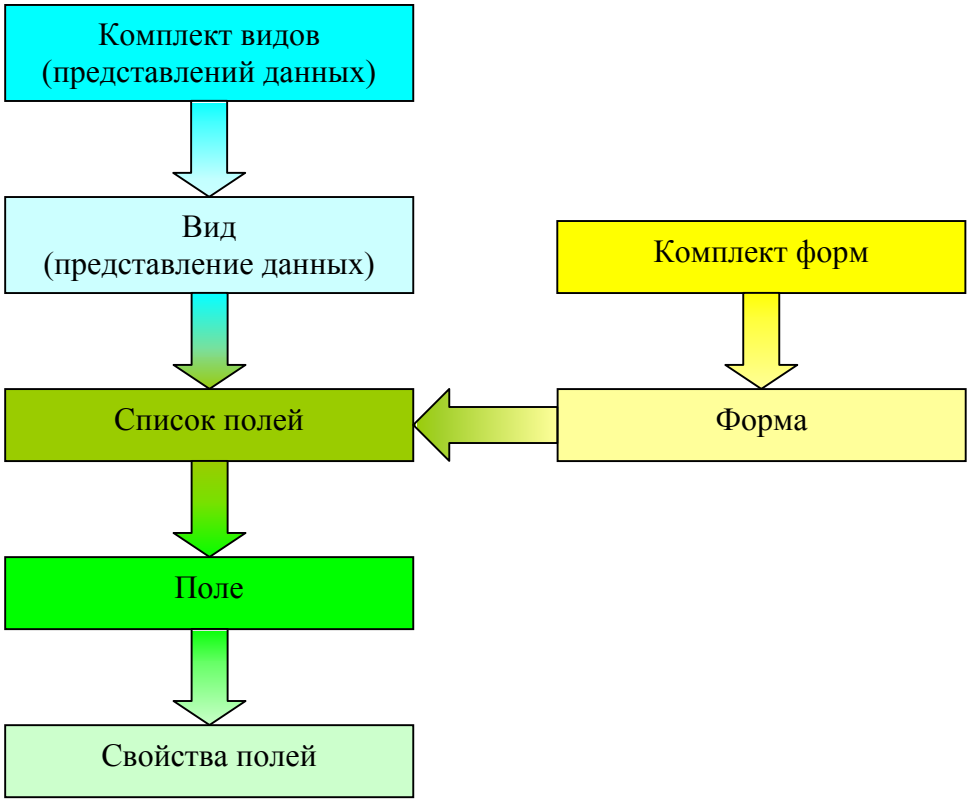

Рисунок 3. Общая концепция дедуктивного метода перехода «от большего – к меньшему».

Сделать иерархическую систему независимую к переходам как снизу доверху, так и сверху донизу инвариантную к направлению и уровням.

Для этого используется концепция «от меньшего к большему» и «от большого к меньшему». По этой концепции сначала дается описание самой малой единицы, и далее доходим последовательно по уровням иерархии до глобальной единицы, получается некоторое подобие библиотеки наоборот.

4. Пути решения задачи

4.1. Принятые определения.

Свойство поля – определяется путем выяснения, данные какого типа, могут быть, занесены.

Таблица 2. Возможные (широкоупотребимые) свойства полей форм.

№	Имя свойства	Примечание
1.	Name	Внутреннее имя, назначаемое элементу управления. Это имя используется для идентификации имени поля при отправке информации на Web-сервер.
2.	Value	Текст, помещаемый в поле по умолчанию.
3.	Checked	Указывает, устанавливается ли переключатель по умолчанию
4.	Size	По умолчанию имеет значение True. Указывает, что по умолчанию выбирается первая позиция в списке.
5.	Selected	Определяет, выбирается ли по умолчанию первая позиция в списке.
6.	Multiselect	Определяет, можно ли выбрать более одной позиции. По умолчанию равен True. При изменении значения параметра MultiSelect отменяются значения параметра Selected.
7.	Maxlength	Максимально возможное число вводимых символов. По умолчанию значение равно 0, что означает отсутствие ограничений.
8.	Columns	Ширина области ввода (число столбцов).
9.	Rows	Высота области ввода (число строк).
10.	Action	Укажите расположение файла, который открывается при нажатии кнопки отправки. Этот параметр становится адресом URL тега <FORM>. В этом поле содержится информация для операций отправки сообщений: введите почтовый адрес в Интернете после mailto:.
11.	Caption	Текст, изображаемый на кнопке.
12.	Method	Метод, используемый для отправки формы: POST или GET.
13.	Encoding	Содержит код MIME, используемый для кодировки подтвержденной формы. По умолчанию в этом поле содержится «application/x-www-form-urlencoded».
14.	Source	Имя исходного файла изображения.

На основании таблицы 2 получаем XML – структуру свойств полей.

```
<?xml version="1.0" encoding="windows-1251" ?>
- <Properties>
    <Propertie id="1" Name="Name" Comment="Внутреннее имя, назначаемое элементу управления. Это имя используется для
      идентификации имени поля при отправке информации на Web-сервер" />
    <Propertie id="2" Name="Value" Comment="Текст, помещаемый в поле по умолчанию" />
    <Propertie id="3" Name="Checked" Comment="Указывает, устанавливается ли переключатель по умолчанию" />
    <Propertie id="4" Name="Size" Comment="По умолчанию имеет значение True. Указывает, что по умолчанию выбирается
      первая позиция в списке" />
    <Propertie id="5" Name="Selected" Comment="Определяет, выбирается ли по умолчанию первая позиция в списке" />
    <Propertie id="6" Name="Multiselect" Comment="Определяет, можно ли выбрать более одной позиции. По умолчанию равен
      True. При изменении значения параметра MultiSelect отменяются значения параметра Selected" />
    <Propertie id="7" Name="Maxlength" Comment="Максимально возможное число вводимых символов. По умолчанию значение
      равно 0, что означает отсутствие ограничений" />
    <Propertie id="8" Name="Columns" Comment="Ширина области ввода (число столбцов)" />
    <Propertie id="9" Name="Rows" Comment="Высота области ввода (число строк)" />
    <Propertie id="10" Name="Action" Comment="Укажите расположение файла, который открывается при нажатии кнопки
      отправки. Этот параметр становится адресом URL тега FORM. В этом поле содержится информация для операций
      отправки сообщений введите почтовый адрес в Интернете после mailto" />
    <Propertie id="11" Name="Caption" Comment="Текст, изображаемый на кнопке" />
    <Propertie id="12" Name="Method" Comment="Метод, используемый для отправки формы: POST или GET" />
    <Propertie id="13" Name="Encoding" Comment="Содержит код MIME, используемый для кодировки подтвержденной формы" />
    <Propertie id="14" Name="Source" Comment="Имя исходного файла изображения" />
  </Properties>
```

Рисунок 4. Пример структурного файла для свойств полей.

Поле – единица заполнения данных, в соответствии с приведенной выше таблицей свойств.

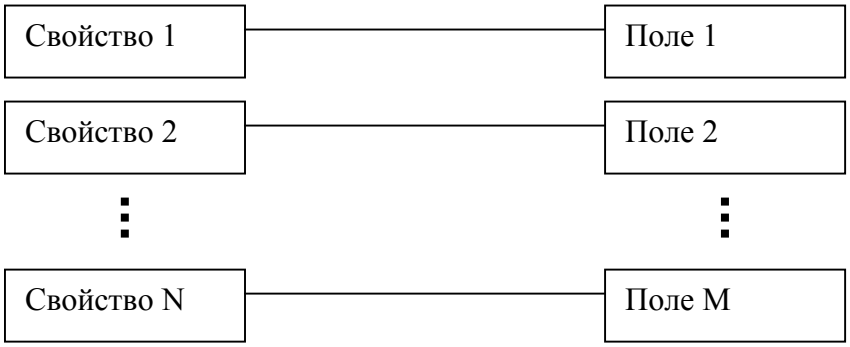

Рисунок 5. Схематичная структура построения взаимосвязи полей и их свойств.

Поля также различаются по возможности внесения в них данных. Рассмотрим таблицу возможных значений полей.

Таблица 3. Возможные поля форм.

№	Наименование	Примечание
1.	Input type = text	Создает элемент для ввода текста из одной строки.
2.	Input type = password	Аналогичен значению "text", но вводимый текст представляется таким образом, чтобы не отображать символы (например, в виде ряда звездочек). Этот управляющий элемент часто используется для ввода паролей. Обратите внимание, что текущим значением является текст, введенный пользователем, а не текст, представляемый агентом пользователя.
3.	Input type = checkbox	Флажки (и кнопки с зависимой фиксацией) - это переключатели вкл./выкл., которые могут переключаться пользователем. Переключатель "включен", если для управляющего элемента установлен атрибут selected.
4.	Input type = radio	Кнопки с зависимой фиксацией похожи на флажки за исключением того, что, если несколько кнопок используют одно и то же имя управляющего элемента, они являются взаимоисключающими: если одна кнопка включена, другие обязательно выключены. Для создания кнопок с зависимой фиксацией используется элемент INPUT.
5.	Input type = submit	Создает кнопку отправки.
6.	Input type = image	Создает графическую кнопку отправки. Значение атрибута src задает URI изображения, используемого для представления кнопки. Из соображений доступности авторам следует предусматривать альтернативный текст для изображения с помощью атрибута alt.
7.	Input type = reset	Создает кнопку сброса.
8.	Input type = button	Создает другую кнопку. Агенты пользователей должны использовать в качестве метки на кнопке значение атрибута value.
9.	Input type = hidden	Создает невидимый управляющий элемент.
10.	Input type = file	Создает управляющий элемент выбор файла. Агенты пользователей могут использовать значение атрибута value в качестве исходного имени файла.
11.	Label	Создает текст метки поля формы
12	textbox	TEXTAREA - элемент для ввода текста из нескольких строк.

Такие поля как checkbox и radio, могут быть многозначными для выбора из массива данных.

На основании таблицы 3 получаем структуру для возможных полей ввода информации.

```
<?xml version="1.0" encoding="windows-1251" ?>
- <Fields>
    <Field Id="1" Name="Input" Type="text" Comment="Создает элемент для ввода текста из одной строки" />
    <Field Id="2" Name="Input" Type="password" Comment="Аналогичен значению text, но вводимый текст представляется таким
      образом, чтобы не отображать символы (например, в виде ряда звездочек). Этот управляющий элемент часто
      используется для ввода паролей. Обратите внимание, что текущим значением является текст, введенный
      пользователем, а не текст, представляемый агентом пользователя" />
    <Field Id="3" Name="Input" Type="checkbox" Comment="Флажки (и кнопки с зависимой фиксацией) - это переключатели вкл.-
      выкл., которые могут переключаться пользователем. Переключатель включен, если для управляющего элемента
      установлен атрибут selected" />
    <Field Id="4" Name="Input" Type="radio" Comment="Кнопки с зависимой фиксацией похожи на флажки за исключением того,
      что, если несколько кнопок используют одно и то же имя управляющего элемента, они являются
      взаимоисключающими: если одна кнопка включена, другие обязательно выключены" />
    <Field Id="5" Name="Input" Type="submit" Comment="Создает кнопку отправки" />
    <Field Id="6" Name="Input" Type="image" Comment="Создает графическую кнопку отправки. Значение атрибута src задает
      URI изображения, используемого для представления кнопки. Из соображений доступности авторам следует
      предусматривать альтернативный текст для изображения с помощью атрибута alt" />
    <Field Id="7" Name="Input" Type="reset" Comment="Создает кнопку сброса" />
    <Field Id="8" Name="Input" Type="button" Comment="Создает другую кнопку. Агенты пользователей должны использовать в
      качестве метки на кнопке значение атрибута value" />
    <Field Id="9" Name="Input" Type="hidden" Comment="Создает невидимый управляющий элемент" />
    <Field Id="10" Name="Input" Type="file" Comment="Создает управляющий элемент выбор файла. Агенты пользователей
      могут использовать значение атрибута value в качестве исходного имени файла" />
  </Fields>
```

Рисунок 6. Пример структурного файла для возможных полей ввода информации.

Форма – набор полей, объединенных каким-либо общим объектом.

Статическая форма – это форма, в которой при заполнении на протяжении долгого промежутка времени не меняется количество полей, и нормативных и нормированных данных – списков.

Динамическая форма – это форма, в которой при заполнении на протяжении короткого промежутка времени могут изменяться количество полей, и нормативных и нормированных данных – списков.

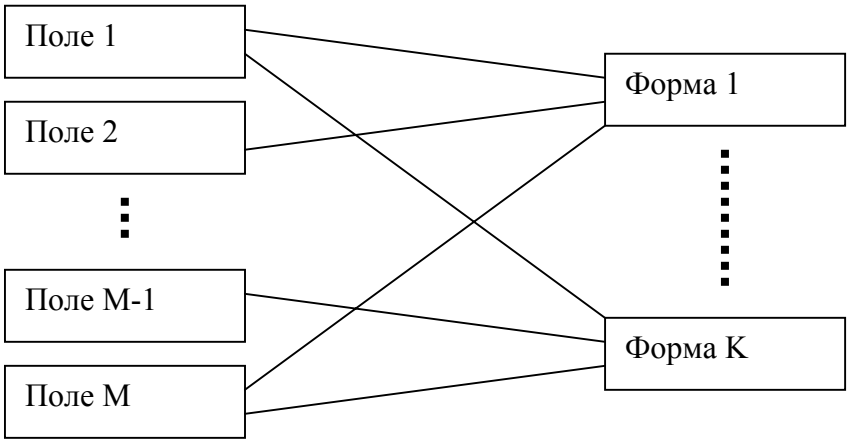

Рисунок 7. Схематичная структура построения взаимосвязи полей и форм.

При этом получаем, что любое поле может принадлежать к одной или нескольким формам, при условии, что поле может входить в группу полей для одной формы, а для другой использоваться отдельно как самостоятельная единица.

Такое же условие распространяется для соотношений между формами и комплектов форм.

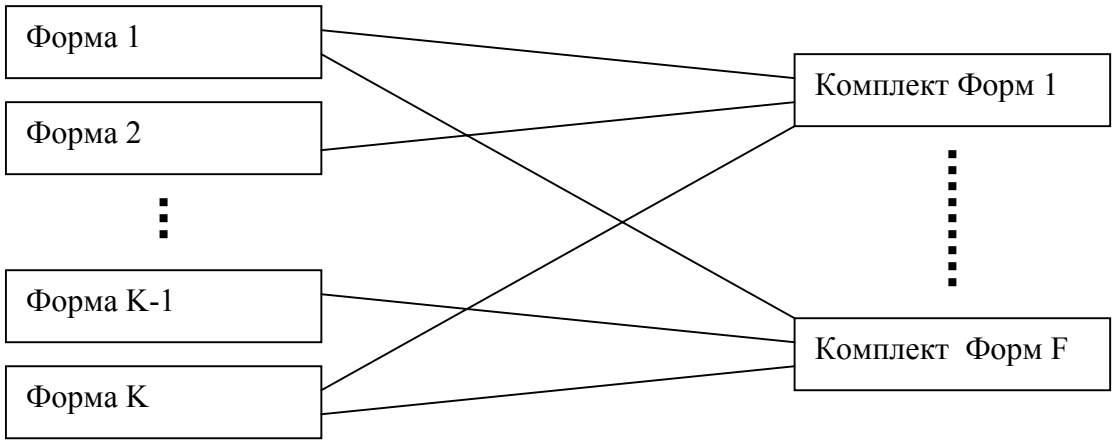

Рисунок 8. Схематичная структура построения взаимосвязи форм и их комплектов.

Комплект форм – набор форм, объединенных глобальных объектов.

На основании вышеприведенных структур можно построить следующую.

```
<?xml version="1.0" encoding="windows-1251" ?>
- <Forms>
  - <StatusForm id="1" Status="Request" StatusRus="Заявка">
      <Form id="1" Number="Форма 1" NameRus="Данные о проекте" NameEng="request" />
      <Form id="2" Number="Форма 2" NameRus="Данные о руководителе и основных исполнителях проекта"
        NameEng="PersonList" />
      <Form id="3" Number="Форма 3" NameRus="Данные об организации" NameEng="OrganizationList" />
    </StatusForm>
  - <StatusForm id="2" Status="Report" StatusRus="Отчет">
      <Form id="4" Number="Форма 1" NameRus="Данные о проекте" NameEng="report" />
      <Form id="5" Number="Форма 2" NameRus="Данные о руководителе и основных исполнителях проекта"
        NameEng="PersonList" />
      <Form id="6" Number="Форма 3" NameRus="Данные об организации" NameEng="OrganizationList" />
      <Form id="7" Number="Форма 6" NameRus="Данные о публикациях" NameEng="PublicationList" />
      <Form id="8" Number="Форма 8" NameRus="Данные о финансировании" NameEng="Estimate" />
    </StatusForm>
  </Forms>
```

Рисунок 9. Пример структурного файла для комплекта форм.

В итоге если существует набор полей можно из него создать Форму, а после создания Форм – перейти к Комплекту Форм, тем самым можно использовать как Поля – созданные конструктором для различных Форм, Комплектов Форм – как стандарт для различных Организаций так как могут не совпадать по нумерации или по представлению Формы, Комплектов Форм.

При концепции «от большего к меньшему», от глобальной единицы до самой маленькой, иллюстрирует прямое подобие библиотеки.

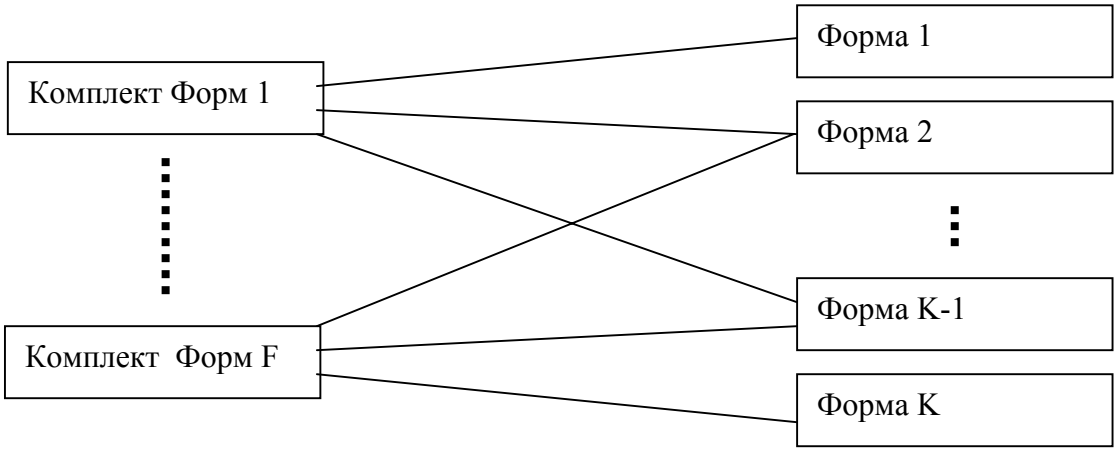

Рисунок 10. Схематичная структура построения взаимосвязи комплектов форм (комплектов представлений, комплектов видов) и информационных представлений, видов.

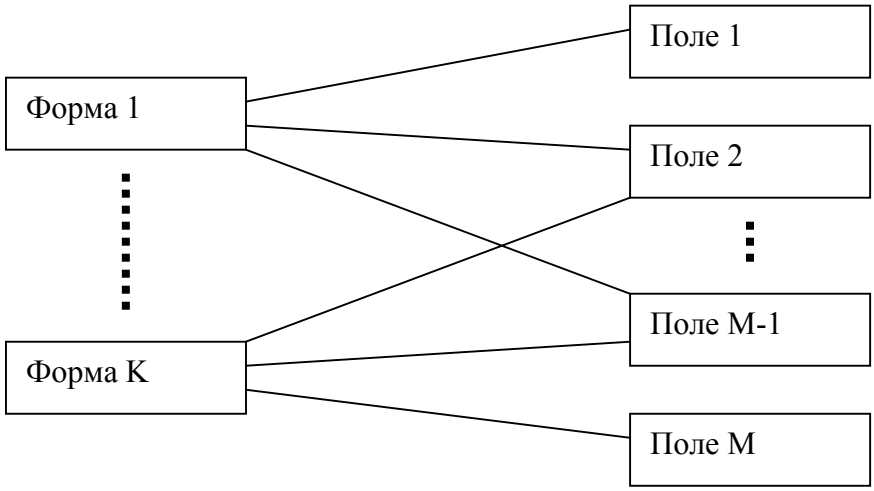

Рисунок 11. Схематичная структура построения взаимосвязи форм (представлений, видов) и полей.

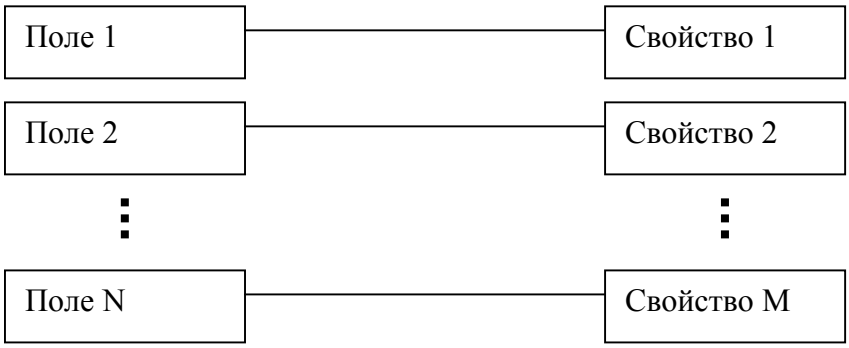

Рисунок 12. Схематичная структура построения взаимосвязи полей и их свойств.

Для этой задачи приходится загружать не только данные, но и сами формы, что приводит к потере быстродействия.

Рассмотрим несколько вариантов работы пользовательского компьютера с веб-сервером:

Веб-сервер – отдельный компьютер, имеющий выход в Интернет, в дальнейшем «Сервер» (Рис. 13).

С точки зрения быстродействия необходимо рассмотреть взаимодействие программного обеспечения внутри компьютера-сервера и пользовательским компьютером, и разработать методику повышения производительности.

1. Программная оболочка веб-сервера – программа, которая отвечает на запрос пользователя о получении информации (Apache, IIS).

2. Обработчик скриптовых языков – программа, которая отвечает за получение (из текстовых файлов, из баз данных) и представлении информации (будь то текстовая, табличная, графическая форма), в удобном для пользователя виде (Java, Microsoft Framework, Perl, PHP).

3. Хранилище данных – способы хранения данных: текстовые файлы, базы данных, структурированные файлы (корпораций Microsoft, Adobe, HTML, XML).

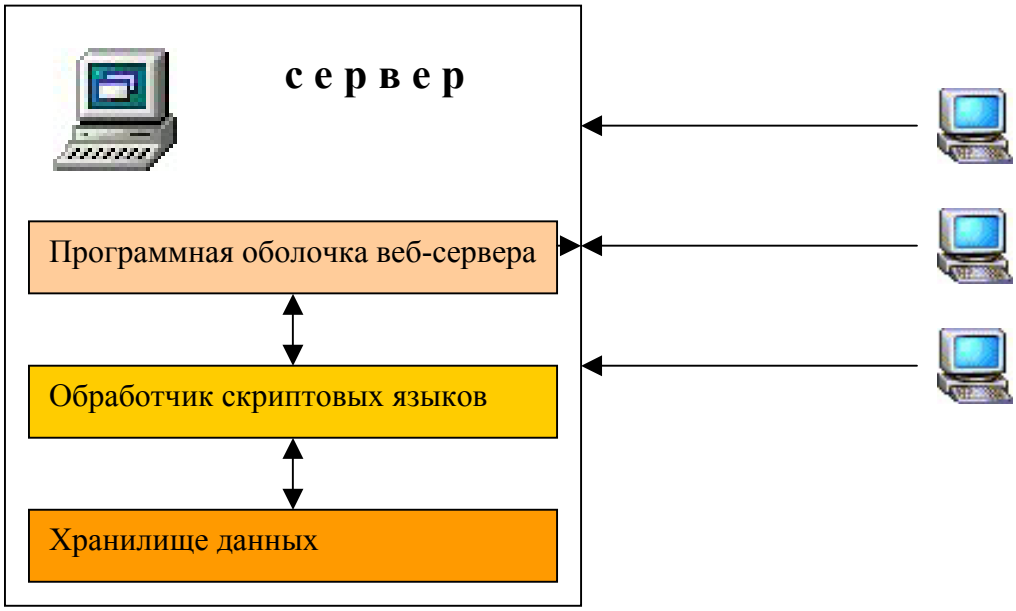

Рисунок 13. Взаимодействие пользователей и программного обеспечения сервера.

Как видно из рисунка, большую роль в быстродействии сервера играет хранилище данных, так как, именно к нему за получением информации обращается обработчик скриптов.

Есть три возможности хранения и обработки комплекта форм научных документов.

1. Статический - основан на том, что любая форма может быть сконструирована один и только один раз, и она записывается в программную оболочку, то есть не может быть изменена без соответствующего программного обеспечения, будь то «Блокнот» или более «продвинутая» программа для редактирования - представления html – страниц.

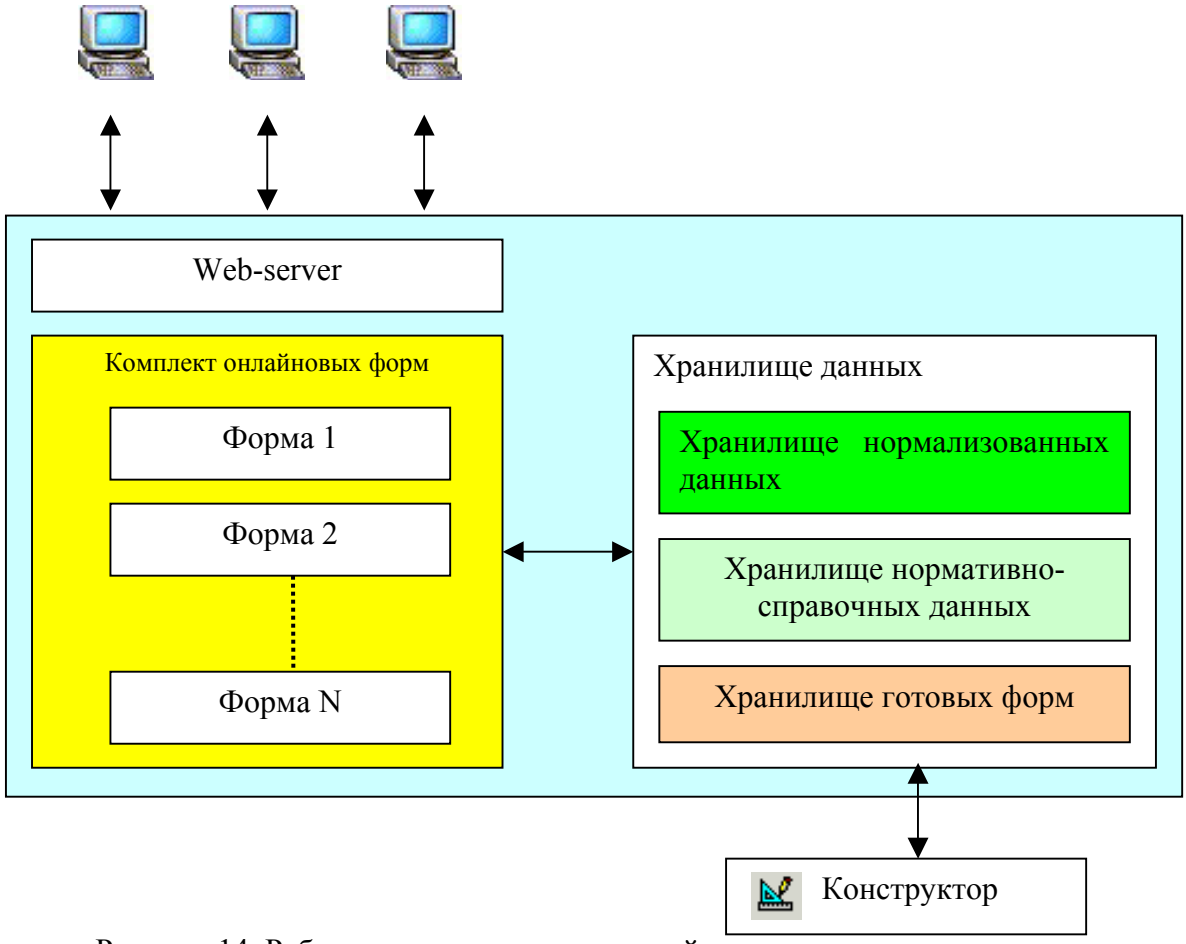

Рисунок 14. Работа сервера и пользователей при статическом варианте.

Особенности:

- Структура нормативно-справочных данных формируются на стадии разработки и, не может быть изменена без вмешательства разработчика (конструктора), но данные могут быть изменены в любой момент времени. Могут быть по структуре не одинаковыми, так как, обработка происходит на уровне конструкционных методов разработчика;

- Структура нормализованных списков – Организаций, Персон и т.п., формируются на стадии разработки и, не может быть изменена без вмешательства разработчика (конструктора), но данные могут быть изменены в любой момент времени. Могут быть по структуре не

одинаковыми, так как, обработка происходит на уровне конструкционных методов разработчика;

- Структура комплекта форм, формы и поля форм для заполнения пользователем формируются на стадии разработки, не может быть изменена без вмешательства разработчика (конструктора);

- Структура представлений (текстовое, табличное, графическое) данных формируются на стадии разработки, не может быть изменена без вмешательства разработчика (конструктора).

В результате получаем систему, спроектированную один раз, без возможности внесения обновлений – дополнений без разработчика.

РЕЗЮМЕ: При первом подходе, происходит загрузка форм уже готовых к работе, пользователь получает нормализованные и нормативно-справочные данные структурированные и описанные разработчиком, поэтому время обращения к такой системе мало.

2. Динамический (конструктивный) - основан на получении всех данных о комплекте форм, самих форм, и их полей, электронного научного документа из конструктора, при этом, структура данных хранятся независимо от программной оболочки, которую использует разработчик.

Особенности:

- Структура нормативно-справочных данных формируются на стадии разработки, но она может быть изменена без вмешательства разработчика (конструктора), так как структура представляет собой структурированный текст. Данные могут быть изменены в любой момент времени. По возможности должны быть одинаковыми по структуре, так как, обработка происходит на уровне одного алгоритма работы с таким видом данных;

- Структура нормализованных списков – Организаций, Персон и т.п., формируются на стадии первоначальной разработки, но она может быть изменена без вмешательства разработчика (конструктора), так как структура представляет собой структурированный текст. Данные могут быть изменены в любой момент времени. По возможности должны быть одинаковыми по структуре, так как, обработка происходит на уровне одного алгоритма работы с таким видом данных;

- Структура комплектов форм, формы и поля форм для заполнения пользователем формируются на стадии разработки, но она может быть изменена без вмешательства разработчика (конструктора), так как представляет собой структурированный вид;

- Структура представлений и их вид (текстовое, табличное, графическое) данных формируются на стадии разработки, но она может быть изменена без вмешательства разработчика (конструктора), так как представляет собой структурированный вид.

При этом варианте, если есть ссылки на нормализованные и нормативно-справочные данные, происходит сначала загрузка структур форм, а потом уже их наполнения. Можно уменьшить время обработки данных, сделав структуру предметной области, для оптимизации метода обращения, данных одинаковой, но это не всегда получается, приходится писать отдельные алгоритмы для обработки каждого списка.

После этого пользователь получает структурированные и описанные разработчиком формы для заполнения, поэтому время обращения к такому хранилищу данных намного увеличивается, при этом играет роль количество и размер нормализованных и нормативно-справочных данных.

РЕЗЮМЕ: Можно вносить любые изменения в любой момент времени, и эти изменения будут касаться не только форм для ввода информации, но и менять структуру самого комплекта, а также видоизменять представления для пользователя (текстовый, табличный, графический).

Получаем проигрыш по времени сравнительно первому варианту.

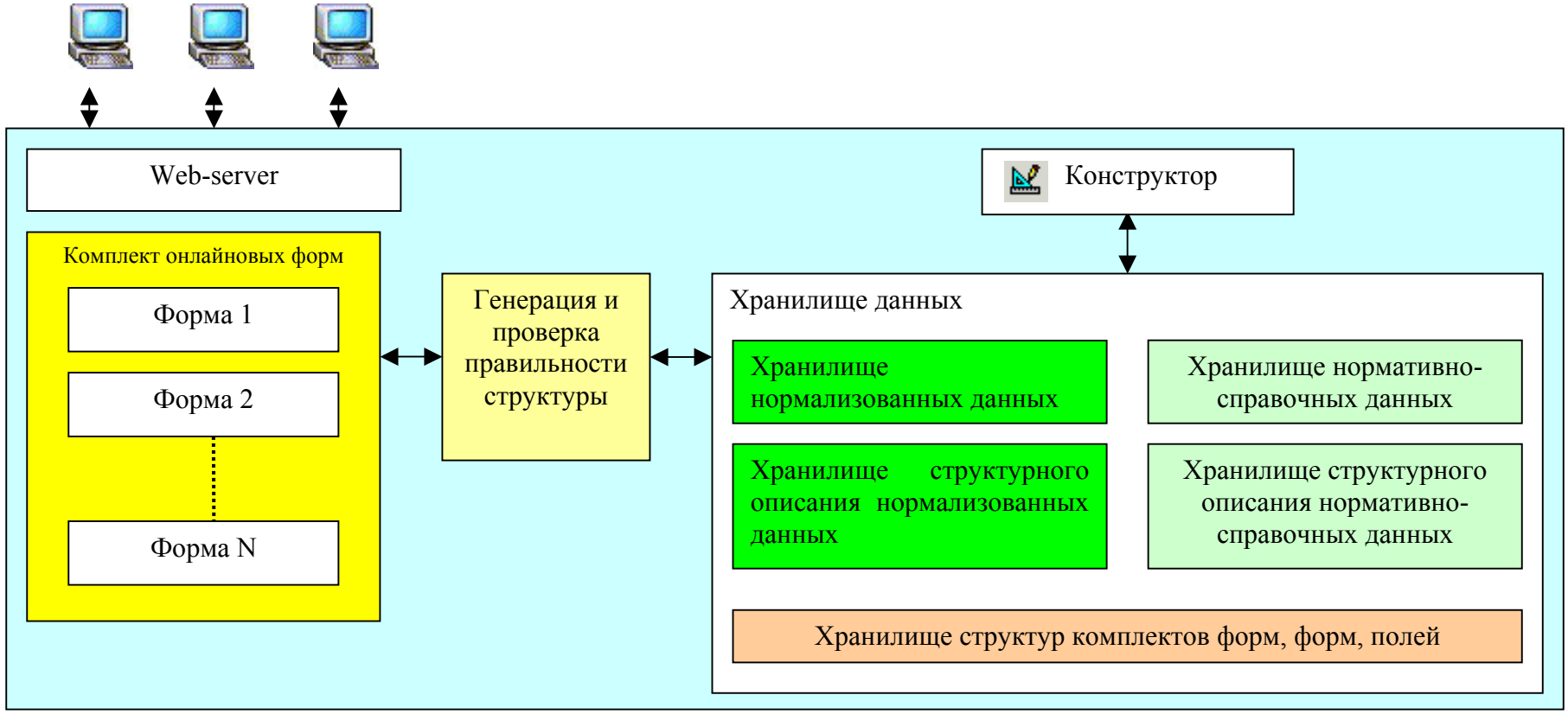

Рисунок 15. Работа сервера и пользователей при динамическом варианте.

3. Комбинированный (статически-динамический) вариант основан на получении всех данных о комплекте форм, самих форм, и их полей, электронного научного документа из конструктора, при этом, данные о структуре данных хранятся независимо от программной оболочки, которую использует разработчик, но в отличии от второго варианта, с помощью конструктора генерируются готовые формы, которые потом хранятся на сервере.

Особенности:

- Структура нормативно-справочных данных формируется на стадии разработки, но она может быть изменена без вмешательства разработчика (конструктора), так как представляет собой структурированный текст. Данные могут быть изменены в любой момент времени. По возможности должны быть одинаковыми по структуре, так как, обработка происходит на уровне одного алгоритма работы с таким видом данных;

- Структура нормализованных списков – Организаций, Персон и т.п., формируется на стадии первоначальной разработки, но она может быть изменена без вмешательства разработчика (конструктора), так как структура представляет собой структурированный текст. Данные могут быть изменены в любой момент времени. По возможности должны быть одинаковыми по структуре, так как, обработка происходит на уровне одного алгоритма работы с таким видом данных;

- Структура комплектов форм, формы и поля форм для заполнения пользователем формируются на первоначальной разработки, но она может быть изменена без вмешательства разработчика (конструктора), так как структура представляет собой структурированный вид;

- Структура представлений данных и их вид (текст, табличное, графическое) формируются на стадии разработки, но она может быть зменена без вмешательства разработчика (конструктора), так как структура представляет собой структурированный вид.

При этом варианте происходит загрузка готовых форм, если есть ссылки на нормализованные и нормативно-справочные данные загружаются, то они тоже будут сгенерированы на уровне конструктора.

Пользователь получает структурированные и описанные разработчиком формы для заполнения, поэтому время обращения в этом случае приблизительно равно времени первого варианта, но при этом не теряется возможность изменения структуры и наполнения данных.

РЕЗЮМЕ: Можно вносить любые изменения в любой момент времени, и эти изменения будут касаться не только форм для ввода информации, но и возможности менять структуру самого комплекта, а также менять вид представления не теряя скорости обработки .

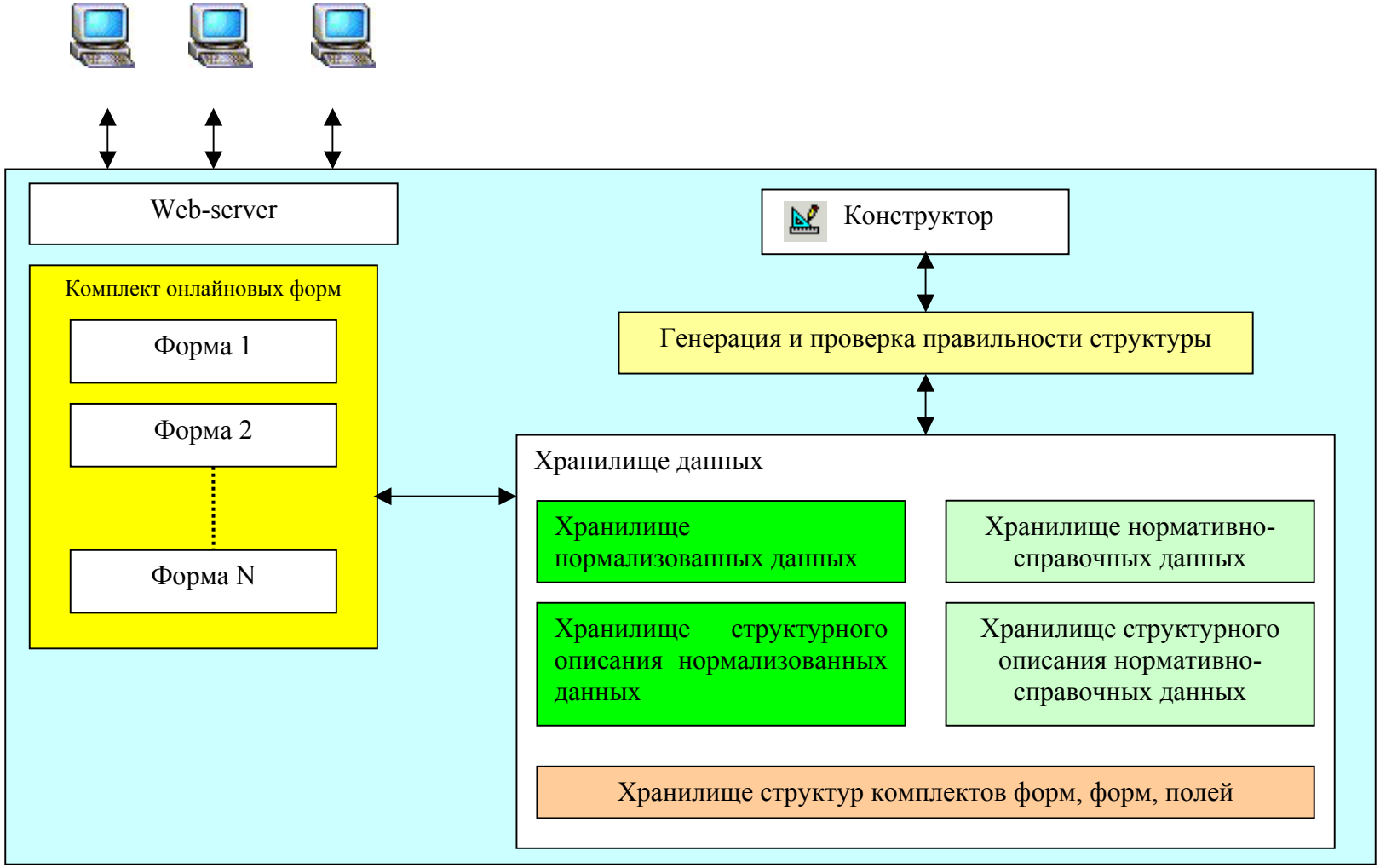

Рисунок 16. Работа сервера и пользователей при комбинированном варианте.

6. Пример практического применения

При индуктивно-дедуктивном методе оказывается удобно создание графиков подобных Рисунку 15 и множества взаимосвязанных таблиц Рисунок 16.

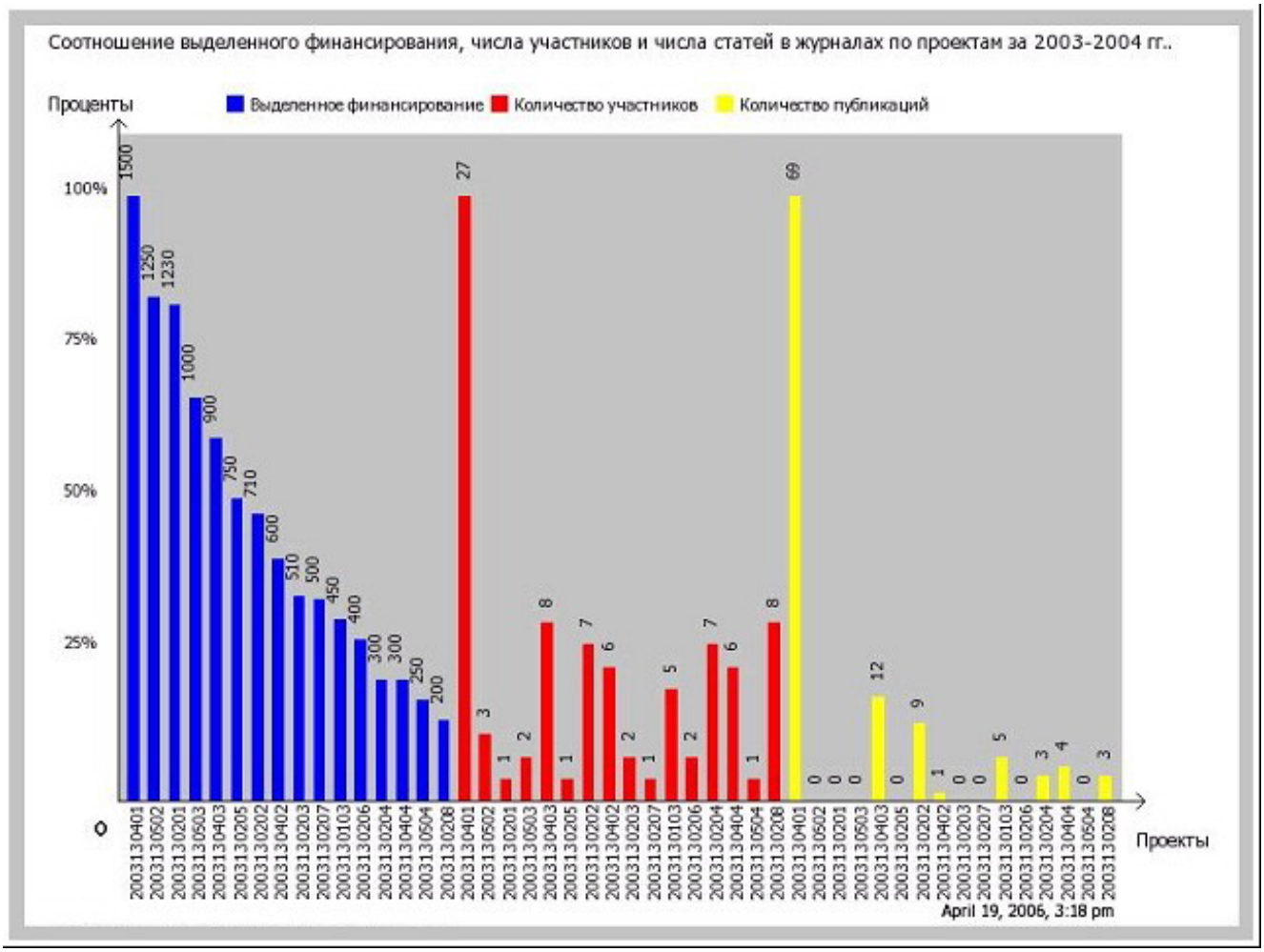

Рисунок 15. Пример практического использования (графический вариант).

Создавая Комплект представлений (видов) надо, прежде всего, указать название «Соотношение выделенного финансирования, числа участников и числа статей», затем указать тип – «графический», затем указать таблицы – формы данных «общая таблица проектов», «таблица финансирования», «таблица участников», «таблица публикаций».

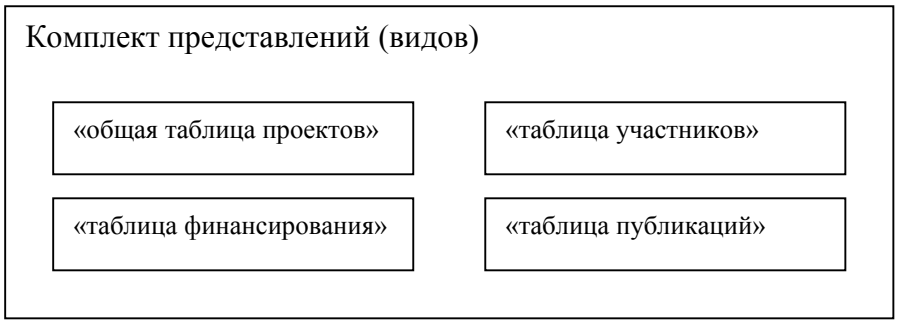

Рисунок 16. Пример практического использования (табличный вариант).

РЕЗЮМЕ:

1.При заполнении научного электронного документа в он-лайн режиме или представлении в табличном или графическом виде используется принцип «От большого – к малому», то есть если существует Комплект Форм (представлений, видов) для Организации Пользователь из выбранной Организации получает доступ только к своему набору Комплекта Форм, соответствующему требованиям Организации и далее по ступенькам происходит заполнение форм - полей.

Заключение.

В работе рассмотрены концептуальные и конструктивные методы построения независимой, иерархической, корпоративной технологии, которая позволяет создавать инвариантные представления и формы данных любого уровня сложности, с длительным сроком службы. Обеспечивается легкость в обращении и построении для всех участников процесса.

Но при этом не следует забывать проблемы, связанные с он-лайн обработкой научных электронных документов:

1. Формирование электронных документов динамически. Для этой задачи приходится загружать не только данные, но и сами формы, что приводит к потере быстродействия.

2. Формирование обновляемых динамических списков. Для этой задачи приходится поддерживать он-лайн соединение на момент создания форм, представлений, видов.

Литература.

1. Alf. Введение в Базы данных. http://club.shelek.com/viewart.php?id=130
2. Кэвин Вильямс. Элементы и Атрибуты: Моделирование Реляционных Данных Средствами XML http://www.realcoding.net/article/view/2760
3. Дон Петерсон. Является ли XML панацеей? http://newsletter.narod.ru/sql_pages/sql_jul_2005.htm
4. Приложение к Положению о порядке разработки планов научно-исследовательских работ научных организаций Российской академии наук