Санкт-Петербургский Государственный Университет
Математико-маханический факультет
Кафедра системного программирования

Проектирование и реализация облачной metaCASE системы

Курсовая работа студентки 445 группы
Чижовой Надежды Александровны

Научный руководитель Литвинов Ю.В.

Оглавление

1. Введение

Тема облачных вычислений становится все популярнее на IT-рынке. Подобные сервисы — новая парадигма создания приложений, которая постепенно вытесняет с рынка традиционные клиент-серверные, многозвенные и распределенные решения в связи с тем, что облака позволяют не заботиться о конфигурации и сбоях оборудования, не требуют мощного компьютера, ускоряют процесс обмена данными. Согласно некоторым прогнозам, практически все популярные программные продукты будут скоро доступны в виде облачных сервисов.

1.1 Облака

Облачные вычисления — технология обработки данных, в которой компьютерные ресурсы и мощности предоставляются пользователю как интернет-сервис.

Сервисы, предоставляемые облачными платформами, бывают трех видов:

- **Iaas** - инфраструктура как сервис. Данная модель предоставляет возможность аренды таких инфраструктурных ресурсов, как серверы, устройства хранения данных и сетевое оборудование. Управление всей инфраструктурой осуществляется поставщиком сервисов, а потребитель управляет только операционной системой и установленными приложениями.
- **Paas** - платформа как сервис. Подобный вариант использования облачных ресурсов предполагает возможность аренды платформы, которая обычно включает операционную систему и прикладные сервисы. Платформа как сервис облегчает разработку, тестирование, развертывание и сопровождение приложений без необходимости инвестиций в инфраструктуру и программную среду. Данная модель также включает в себя и инфраструктуру как сервис.
- **Saas** - программное обеспечение как сервис. “Программное обеспечение как сервис” является наиболее распространенной на сегодняшний день моделью предоставления облачных сервисов. Эта модель включает в себя две другие, т.е облачная платформа используется для хостинга приложений, хранения данных и

проведения вычислений. Данная модель является наиболее распространенной на сегодняшний день моделью предоставления облачных сервисов.

1.2 MetaCASE системы

При проектировании сложных программных продуктов распространено использование CASE систем.

Если термин CASE (Computer Aided Software Engineering) понимать буквально, то CASE-система - средство, помогающее в разработке программного обеспечения. Таким образом транслятор или любая система программирования является CASE-системой. Однако системы, первоначально появившиеся на рынке программных продуктов под названием CASE - это программные продукты, поддерживающие этап анализа проектируемой системы и фиксацию результатов этой работы в виде соответствующих спецификаций.

Однако за последнее время требования к CASE-средствам возросли. Подобные системы теряют свою полезность, если трудоемкость их разработки достаточно высока.

В связи с необходимостью упростить процесс создания CASE систем появилось понятие metaCASE систем. Данные программные продукты являются CASE средствами для создания CASE систем. Подобные программы значительно упрощают процесс создания CASE систем. MetaCASE системы позволяют по описанию языка автоматически генерировать визуальные редакторы, генераторы и другие средства инструментальной поддержи. Как правило, описание языка задается в виде метамодели.

2. Постановка задачи

Облачные вычисления особенно выгодны для небольших компаний, которые имеют незначительные ресурсы и для которых практически главную роль в принятии того или иного решения играет стоимость разработки и последующей эксплуатации приложения.

Однако, использование облака далеко не единственный способ значительно снизить затраты на разработку и подержку IT-решений. Популярным способом снижения стоимости разработки программного продукта является использование CASE - средств. В связи с чем появилась мысль о создании metaCASE системы, существующей в облаке.

В рамках данной курсовой работы требовалось разработать архитектуру подобной системы и реализовать ее серверную часть. Перед проектировании архитектуры необходимо было разработать технические требования, и формат передачи данных метамодели между клиентом и сервером. Также предполагалась апробация полученной архитектуры на примере редактора приложений для мобильных телефонов. Для этого необходимо было совместить разработанную мной серверную часть, часть клиента и компоненту, отвечающую за генерацию приложения по полученным от клиента данным.

3. Решение поставленных задач

Использованее облачных платформ значительно снижает стоимость разработки программных продуктов. Необходимо оплачивать исключительно ресурсы, выделяемые облаком для работы приложения.

Предоставляемые облачными платформами ресурсы делятся на:

- Аппаратные - облачное хранилище, сервера, компьютерные мощности.
- Коммуникации - каналы связи и передающийся по ним трафик.

В связи с данной особенностью одно из основных требований к архитектуре облачного приложения - минимизация использования каналов связи для передачи данных. Для этого, при проектировании подобных систем, активно работающий с данными код располагают максимально близко к хранилищу, без использования дополнительных слоев кода между ними.

Данная особенность была учтена при разработке архитектуры:

- Компонента системы, отвечающая за генерацию кода, работает напрямую с облачным хранилищем, не используя посредников.
- Для уменьшения трафика, передаваемого по каналам связи, на настоящий момент, клиент посылает данные на сервер только один раз, после формирования в виде xml документа, построенной пользователем диаграммы.

3.1 Средства разработки

В качестве среды разработки был выбран Eclipse Classic 3.7.2. Данная среда предоставляет множество различных средств для работы с языком Java.

В качестве облака была выбрана облачная платформа для хостинга java-приложений

Jelastic. Данная платформа является удобной хостинг-платформой для Java с возможностью запуска и масштабирования любых приложений без необходимости изменения кода.

Отличительные особенности платформы Jelastic:

- поддержка стандартного стека технологий;
- визуальный конструктор топологий;
- вертикальное масштабирование;
- широкий набор баз данных и серверов приложений;
- возможность собирать проекты прямо на платформе. При использовании в проекте таких инструментов, как, например, maven, для корректной работы приложения в облаке достаточно отредактировать конфигурационные файлы. Собирать проект, используя локальный компьютер, не требуется.

· репликация сессий и баз данных.

Описанная облачная платформа поддерживает реляционые базы данных MariaDB, MySQL, PostgreSQL и сервера приложений Jetty 6, Tomcat 6, Tomcat 7, GlasssFish 3. Для максимально рационального использования ресурсов облака в рамках данной курсовой была выбрана база данных MySQL, как наиболее распространенная и простая в использовании, и контейнер сервелетов Jetty 6, как наимение ресурсо-затратный.

Количество потребляемых контейнером сервелетов ресурсов при использовании данного решения не превышает 300-310Мб (5 claudlet), где claudlet - единица измерения ресурсов, используемая в Jelastic. Claudlet - примерный эквивалент 128мб оперативной памяти и 200Мгц процессорной мощности.

Также в рамках данной курсовой был использован контейнер сборки проектов Maven.

3.2 Требования к архитектуре

1. Функциональные требования

Требования к архитектуре:

- Выдерживать сбои и восстанавливаться после них.
  Внутренние сбои или отказы компонентов программы не должны отражаться на клиенте. Система должна уметь самостоятельно восстанавливаться после критических ситуаций.
- Хранить хотя бы часть базы в облачном хранилище.
  База данных системы должна храниться частично в облачном хранилище, частично на клиенте или локальных дисках. Ни в коем случае только на клиенте и локальных дисках.
- Возможность сохранения результата, созданного пользователем.
  Необходимо обеспечить пользователю возможность прерывать работу и возобновлять, используя сохраненный результат.
- Минимизировать использование интернет каналов. Интенсивно работающий с данными код должен храниться рядом с базой данных.

Требования к CASE-системе:

- Уметь ссылаться на объекты (с помощью отношений).
  Объекты, создаваемые пользователем, должны быть связаны между собой, с помощью отношений.
- Уметь изменять свойства существующих объектов и отношений.
  Необходимо уметь изменять свойства уже существующих объектов и связывающих их отношений.
- Уметь удалять объекты и отношения.
- Позволять создавать на экране диаграммы, составленные из связанных объектов и

отношений.

Пользователь перетаскивает на экран мышкой объекты и отношения, необходимые для конфигурации его приложения, или, при некоторых условиях, они генерируются автоматически.

2. Нефункциональные требования

- Простота использования.
  Незнакомый с программированием человек после прочтения инструкции должен быть способен написать несложное приложение.
- Устойчивость к сбоям.
  Сбои системы должны быть как можно более незаметны для пользователься.
- Доступность.
  Уровень доступности системы должен быть не меньше 99.9% времени.
- Расширяемость.
  Необходимо обеспечить возможность масштабирования системы, для дальнейшего развития.
- Слабое связывание компонент.
  Сбои одной компоненты системы должны как можно меньше влиять на работу других компонент.
- Реализация интерфейсов для различных браузеров.
  Интерфейс системы не должен зависеть от браузера пользователя.

3. Ограничения

- Использование Jelastic.
- Интерфейс на HTML5.

3.3 Формат обмена данными

В задачах построения информационных систем одной из основных проблем является обмен данными между различными компонентами подсистемы. В рамках моей курсовой в качестве формата передачи данных было решено использовать язык разметки XML . При создании собственного языка разметки существует возможность придумывать практически любые названия элементов, что делает использование подобных языков достаточно удобным.

По своей структуре документ в формате XML представляет собой дерево элементов. Некоторые элементы имеют свои дополнительные атрибуты, который в свою очередь могут иметь какие-то значения. Основным требованием, предъявляемым к формату обмена данными, являлась расширяемость протокола. XML, в отличие от других возможных форматов передачи данных, например, таких как JSON, идеально удовлетворяет данному требованию благодаря простой древовидной структуре. В случае возникновения новых свойств, добавление новых узлов в передаваемый формат не представляет сложности. Также в подобном случае, в связи с древовидной структурой формата, нет необходимости изменять вручную код приложения, связанный с разбором xml.

Важным преимуществом подобного языка разметки является простота разбора на сервере и клиенте с помощью средств стандартной библиотеки java.

Пример разработанного формата XML документа для передачи элемента типа “Прямоугольник”:

```
<node name="Rectangle">
        <graphics>
          <picture sizex="200" sizey="200">
            <rectangle fill="#ffffff" stroke-width="0"/>
            <line y1="0" x1="20" y2="0" x2="180"/>
            <line y1="200" x1="20" y2="200" x2="180"/>
          </picture>
```

```
            <ports>
                <pointPort x="20" y="100"/>
                  <pointPort x="80” y="0"/>
                   <pointPort x="80" y="200"/>
                    <pointPort x="180" y="100"/>
                <pointPort x="80" y="100"/>
                <linePort>
                   <start startx="180" starty="0"/>
                   <end endx="20" endy="200"/>
                </linePort>
                <linePort>
                   <start startx="9" starty="34"/>
                   <end endx="76" endy="35"/>
                </linePort>
            </ports>
        </graphics>
        <logic>
              <properties>
              <property name="Rectangle"/>
            </properties>
        </logic>
    </node>
```

Пример разработаного формата XML документа для передачи элемента типа “Линия”:

```
<node name="Line">
        <graphics>
            <picture sizex="107" sizey="38">
                <rectangle fill="#ffffff" stroke-style="solid"/>
```

```
            <line y1="0" x1="300" y2="0" x2="500"/>
        </picture>
        <ports>
            <pointPort x="0" y="450"/>
            <pointPort x="0" y="300"/>
            <pointPort x="0" y="500"/>
            <linePort>
            </linePort>
        </ports>
    </graphics>
    <logic>
        <properties>
            <property name="Line"/>
        </properties>
    </logic>
</node>
```

В приведенном примере используется разработанная в рамках курсовой система тегов и атрибутов. Полный список тегов и атрибутов с описаниями их значений следующий:

<node>--- Узел на диаграмме, имеет атрибут name (имя узла, как оно должно отображаться в палитре).

Сыновья: тэги graphic и logic

<graphics> --- Описание графической составляющей узла диаграммы (форма, цвет и т.д)

Сыновья: тэги picture, ports, labels

<logic> --- Описание логической составляющая узла диаграмы (связи, свойства)

Сыновья: тэги container, properties

<picture> --- Описание графической отрисовки узла диаграммы в редакторе. Имеет атрибуты:

sizex – размер прямоугольника, в который

можно положить картинку по x;

sizey - размер прямоугольника, в который можно положить картинку по y.

Сыновья: line и rectangle

<rectangle> -- Описание прямоугольника, в который заключен узел диаграммы. Имеет атрибуты:

fill --- цвет заливки (чаще всего – цвет фона);

stroke-width --- ширина линии контура прямоугольника

<line> --- Описание графической отрисовки линии. Имеет атрибуты координат точек начала линии x1, y1 и конца линии - x2, y2.

<ports> --- Описание связей между элементами, которые визуально представимы в виде дуг. С помощью различных комбинаций портов - точек и портов - линий имеется возможность задавать различные варианты связей между узлами диаграммы.

Сыновья: <pointPort> и <linePort>.

<pointPort> --- Порт-точка. Имеет атрибуты координат точки (x,y).

<linePort> --- Порт-линия.

Сыновья:

тэг <start> с атрибутами startx и starty задает координаты точки начала линии.

тэг <end> с атрибутами endx и endy задает координаты точки конца линии.

<container> --- Описание связей между элементами, которые невозможно визуально представить в виде дуг.

Сыновья : тэг <contains x>, где x – название узла диаграммы.

<properties> - Описание свойств элемента.

Сыновья: тэги <property> с атрибутами:

type – тип свойства;

значение свойства (например, для описания имени, значение задется типом name).

4. Репозиторий

Наиболее важная и сложная в реализации составляющая разработанной архитектуры - репозиторий. Необходимо было выбрать подходящий способ хранения информации. Плохо разработанный репозиторий мог замедлить работу веб-приложения.

При разработке архитектуры репозитория необходимо было учесть следующее:

1. Возможность быстро получить все существующие элементы в палитре конкретного языка.
2. Возможность блокировать для остальных пользователей один или несколько элементов диаграммы, используемые кем-то в данный момент.
3. Возможность быстро получить элемент со всеми его свойствами.

Как уже говорилось, в качестве репозитория было решено использовать реляционную базу данных, а именно MySQL 5.0.

В реляционных базах данные хранятся в таблицах. Необходимо было придумать, как хранить в таблицах диаграммы, создаваемые пользователем.

В связи с необходимостью показывать на клиенте палитру всех возможных элементов для выбранного пользователем языка моделирования, все существующие элементы хранятся в таблице NameTable вместе со своим уникальным идентификатором и атрибутом принадлежности тому или иному языку.

Подобная таблица позволяет ускорить доступ к элементам палитры. Для получения названий всех элементов палитры определенного языка достаточно выполнить SQL-запрос:

```
select NameTable.name from NameTable
where NameTable.idlen = res
```

где res - значение идентификатора для выбранного пользователем языка.

Также, для быстрого отображения на клиенте элементов палитры, в репозитории существуют таблицы с названиями, соответствующими элементам палитры и атрибутами, соответствующими возможным атрибутам данного элемента.

После создания пользователем диаграммы на клиенте формируется XML документ и передается серверу. Переданный формат разбирается на сервере и в таблицы с необходимыми именами добавляются новые строки, заполненные значениями соответствующих атрибутов элемента. Также заполняется атрибут принадлежности элемента к определенной диаграмме, который используется компонентой, отвечающей за генерацию кода по созданной пользователем диаграмме.

Таким образом, после получения данных об экземпляре элемента типа “Прямоугольник”, в репозитории создается строка в таблице “Rectangle” со значениями атрибутов:

sizex (INTEGER) = 200, sizey (INTEGER) = 200,
fill (VARCHAR(255)) = #ffffff, stroke-style (INTEGER) = 0,
x01 (INTEGER) = 20, y01 (INTEGER) = 0,
x02 (INTEGER) = 180, y02 (INTEGER) = 0,
x11 (INTEGER) = 20, y11 (INTEGER) = 200,
x12 (INTEGER) 180, y12 (INTEGER) = 200,
px1 (INTEGER) = 20, py1 (INTEGER) = 100,
px2 (INTEGER) = 80, py2 (INTEGER) =0,
px3 (INTEGER) = 80, py3 (INTEGER) = 200,
px4 (INTEGER) = 180, py4 (INTEGER) = 100,
px5 (INTEGER) = 80, py5 (INTEGER) = 100,
startx (INTEGER) = 180, starty (INTEGER) = 0,
endx (INTEGER) = 20, endy (INTEGER) = 200,
startx1 (INTEGER) = 9, starty1 (INTEGER) = 34,
endx1 (INTEGER) = 76, endy1 (INTEGER) = 35,

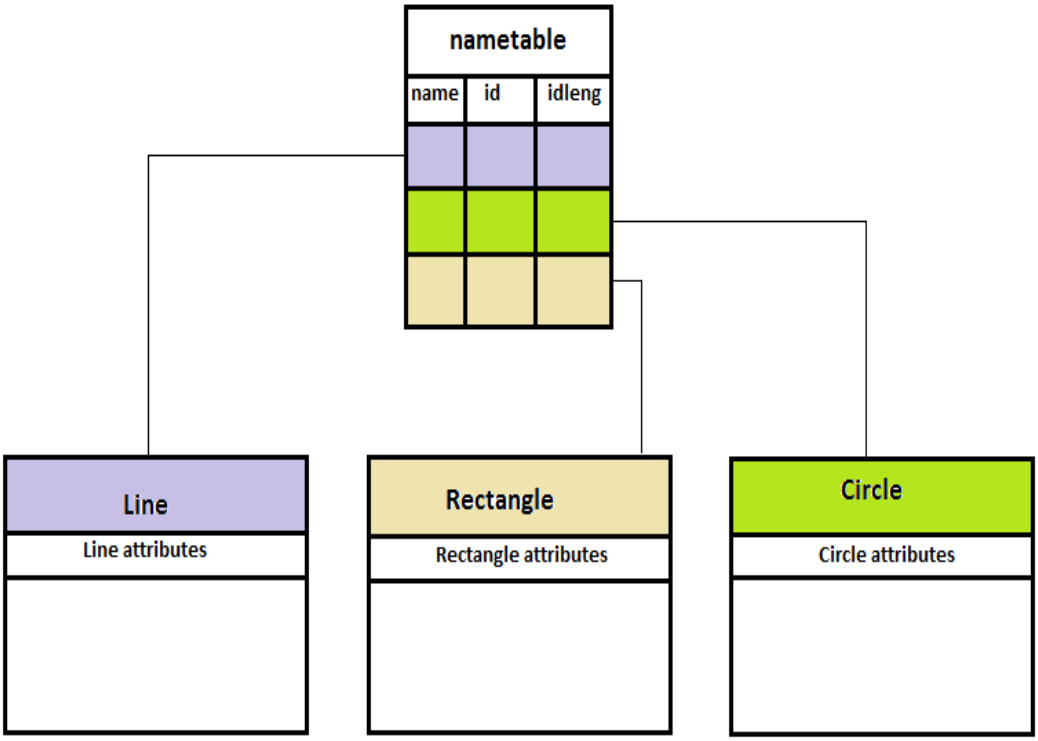
Где все аттрибуты выписаны в порядке появления в формате xml. Таким образом,

например, px1, py1 и далее - все координаты точек-портов данного экземпляра элемента типа Rectangle.

В структуре разработанного репозитория диаграммы, созданные пользователем, хранятся в виде набора строк таблиц, соответствующих типам элементов, из которых данная диаграмма состоит.

4.1 Структура репозитория

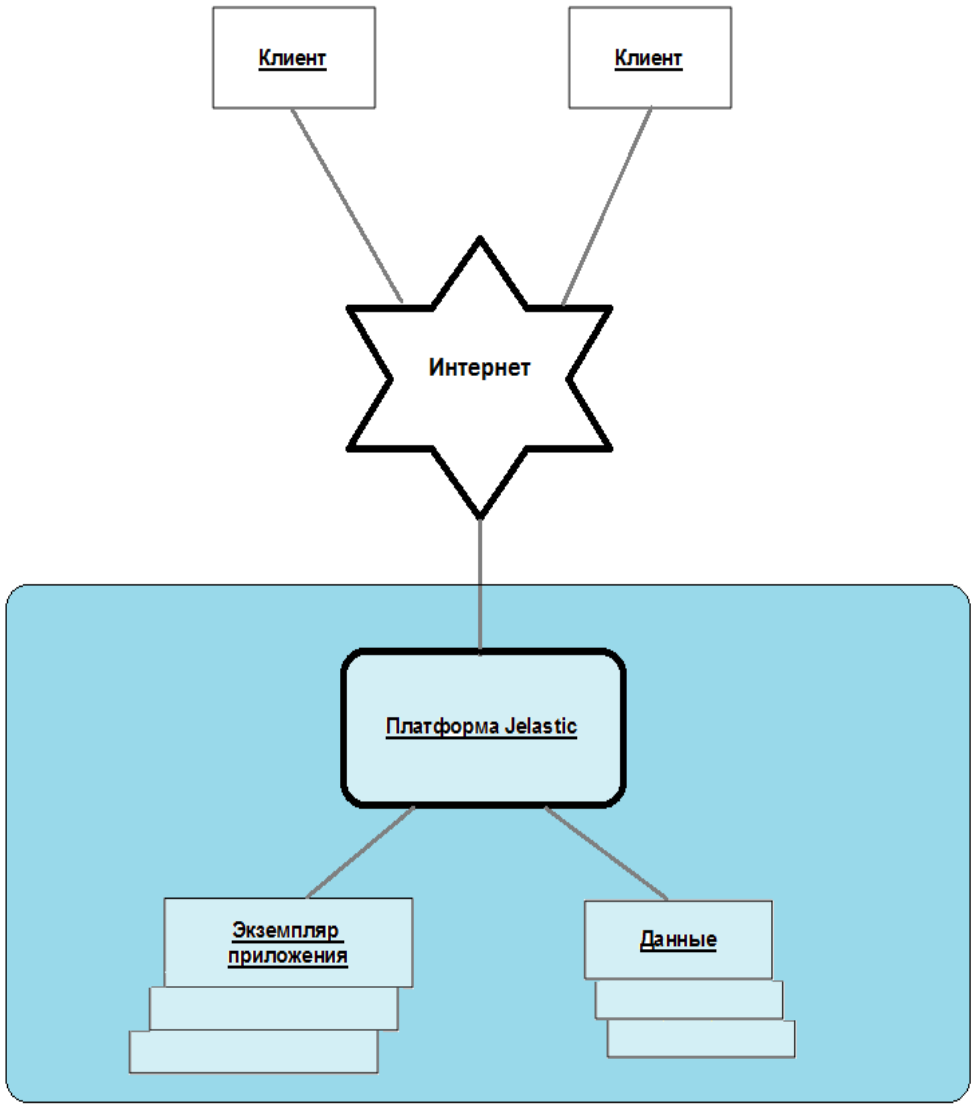
Структура разработанного репозитория выглядит следующим образом:



5. Заключение

5.1 Общая архитектура облачных приложений.

Общая архитектура приложения при использовании облачной платформы выглядит следующим образом:



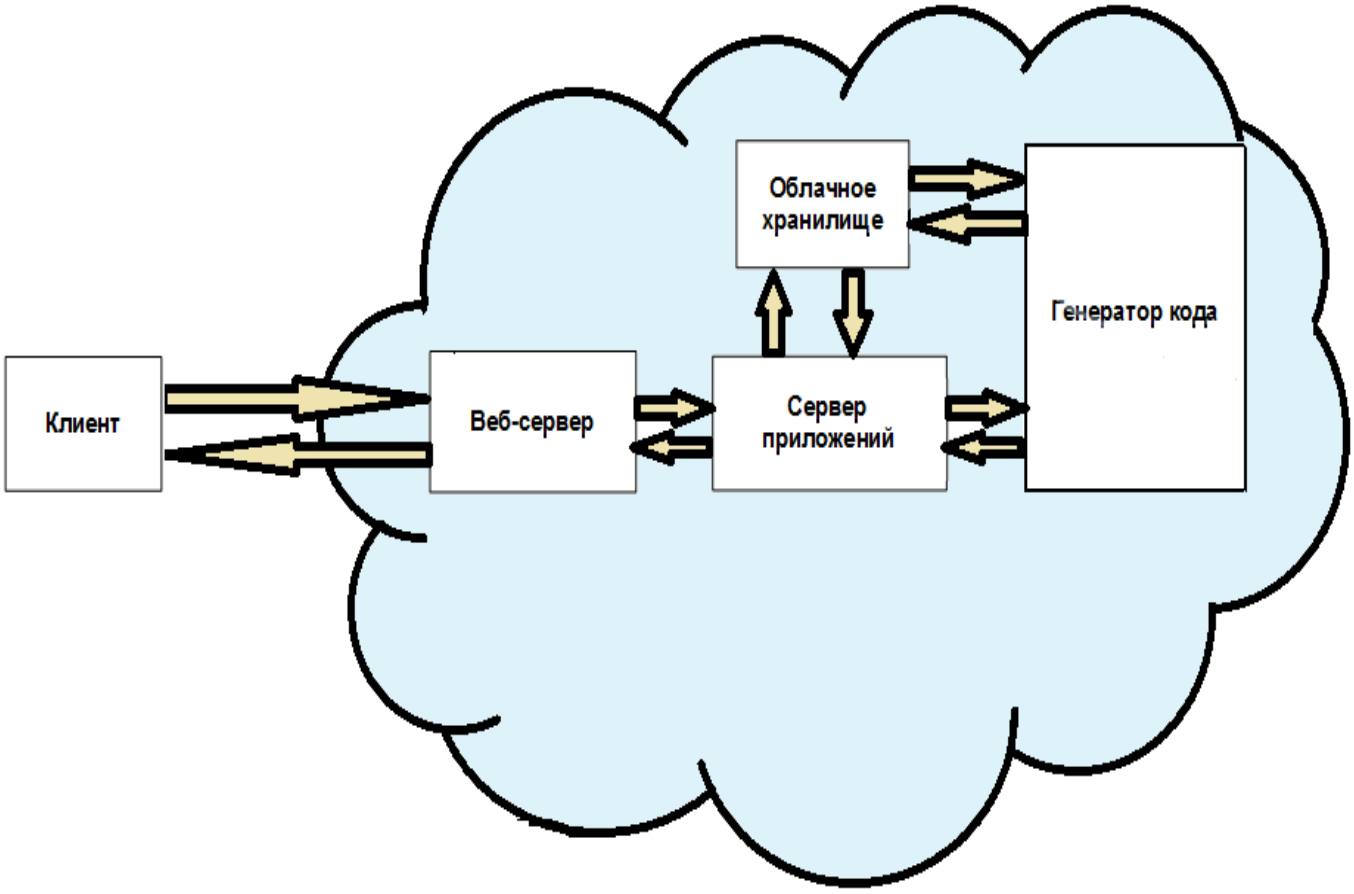
В зависимости от количества пользователей облачная структура выделяет

приложению дополнительные сервера и другие необходимые программе в данный момент ресурсы.

Описанная концепция позволет при проектировании архитектуры не беспокоиться о сбоях оборудования. Подобные сбои контролируются облаком и не отражаются на пользователе.

5.2 Разработанная архитектура

Архитектура разработанной в рамках курсовой работы системы выглядит следующим образом:



Для максимально рационального использования облачной платформы на

данный момент на клиенте только формируется структура формата xml для передачи на сервер, все остальные действия происходят на сервере: полученная xml разбирается и соответствующим образом записывается в базу данных. При необходимости компоненте, отвечающей за генерацию кода, посылается сигнал о возможности взять необходимую информацию из базы (в передаваемом сигнале хранятся, например, идентификационные номера таблиц для элементов, из которых состоит переданная клиентом в формате XML диаграмма). После успешной генерации, сформированные данные (приложение) передаются с помощью сервера приложений обратно на клиент, где, в конечном итоге, выводятся пользователю в качестве результата.

5.3 Возможности для развития

Посколько существует цель уменьшить использование интернет каналов может показаться, что при проектировании облачных приложений следует создавать достаточно “толстый клиент”, на котором будет производиться объемная часть работы и только затем он будет передавать даные на сервер. При таком подходе запросы между клиентом и сервером происходят достаточно редко и каналы связи используются мало.

Однако в таком случае теряются некоторые достоинства при использовании облачной платформы: быстрый обмен данными, отсутствие требований на мощность компьютера. Также возникает проблема с отказоустойчивостью и масштабируемостью, в случае возникновения проблем и ошибок на “толстом” клиенте.

Более глубокий анализ данного вопроса и корректировка соответствующим образом архитектуры планируется в дальнейшем.

Также в последствии предполагается:

- Обеспечение возможности многопользовательской работы.
- Совершенствование структуры репозитория.
- Более четкое разделение функциональности между клиентом и сервером:
    - Проверка синтаксических правил.

Список литературы

1. Алексей Федоров, Дмитрий Мартынов. **Windows Azure. Облачная платформа Microsoft. /** 2010.
http://www.microsoft.com/ru-ru/cloud/tools-resources/whitepaper.aspx?resourceId=cloud_platform

2. Алексей Федоров, Наталия Елманова/ Статья от июня 2002 г: **Архитектура современных web - приложений./** КомпьютерПресс июнь 2002.
http://www.compress.ru/article.aspx?id=10951&iid=440

3. Кузнецов С. Д. **Общая классификация архитектур информационных приложений.//** Проектирование и разработка корпоративных информационных систем. Центр информационных технологий 1998
http://citforum.ru/cfin/prcorpsys/index.shtml

2. Никандров Г. **Дипломная работа: Реализация кроссплатформенной архитектуры CASE-пакета.//** 2007
https://github.com/qreal/qreal/wiki/nikandrov.pdf

5. Татьяна Валентинова/ Статья от 09.03.2009: **Что в действительности представляют собой облачные сервисы.**
http://www.hwp.ru/articles/CHto_v_deystvitelnosti_predstavlyayut_soboy_oblachnie_servisi/

6. Мартин Фаулер, Дейвид Райс, Мэттью Фоммел, Эдвард Хайет, Роберт Ми, Рэнди Стаффорд. **Patterns of Enterprise Application Architecture.** // Addison Wesley 11.05. 2010

7. http://jelastic.com/ru/docs **Документация Jelastic.**

8. http://www.mysql.ru/docs/man/ **MySQL Manual.**

9. http://wiki.eclipse.org/Jetty/ **Jetty Documentation Index @ eclipse.**