# Глава II

# Проектирование сервиса параллельной разработки web-приложений

# Проектирование сервиса параллельной разработки web-приложений

## 1.Стратегия и модель конструирования сервиса

По проведенным исследованиям, компании, использующие при разработке своих продуктов гибкие стратегии, более успешны, чем компании, использующие стратегии однократного подхода.

В связи с этим, в качестве модели конструирования программного средства, было решено использовать эволюционную стратегию, так как она позволяет специфицировать требования к программному продукту в результате разработки его отдельных версий.

Данная модель является наиболее гибкой при разработке программного обеспечения, позволяет разработать небольшое ядро программы, на которое впоследствии можно наращивать функциональность, переходя от версии к версии.

В качестве модели проектирования было решено использовать модель экстремального проектирования.

На данный момент модель экстремального программирования широко популярна. Она позволят разделить планируемые процессы на небольшие кусочки – это приводит к тому, что выпуск первых наработок занимает намного меньше времени, чем в модели водопад.

Сам процесс разработки программного средства завязан на тестировании. Сначала разрабатываются тесты, что бы описать цели разработки, а потом идет кодирование, которое заканчивается в тот момент, когда все тесты пройдены. После того, как закончена работа над данной ступенью разработки программного средства, происходит переход к следующей ступени.

Осуществление разработки через тестирование позволяет отслеживать работоспособность системы на всех ступенях её разработки.

В качестве способа реализации был выбран объектно-ориентированный подход. На данный момент объектно-ориентированное программирование наиболее популярно и обладает более широким спектром возможностей в сравнении с другими парадигмами программирования. Проектирование, кодирование и поддержка программ, разработанных с использованием объектно-ориентированной парадигмы намного легче, чем при использовании процедурного подхода. Возможность повторного использования кода, абстрагирование, наследование, кодогенерация и другие возможности ООП существенно уменьшают время, затрачиваемое на разработку программного обеспечения.

## 2. Предварительное проектирование

### 2.1.Структурирование системы

Так как архитектура данного приложения клиент-серверная, то удобно структурировать клиентскую и серверную части отдельно:



Ядро системы отвечает за загрузку и инициализацию всех подсистем. Обеспечивает передачу сообщений между подсистемами, а так же между клиентской и серверной частями.

Подсистемы клиентской части выполняют действия, свзяанные с подачей информации и предоставлением сервисов по взаимодействию с пользователем.

Каждая подсистема выполняет четко определенные функции и не взаимодействует с другими подсистемами непосредственно. Если одной подсистеме необходимо получить данные, за обработку которых отвечает другая подсистема, то взаимодействие между ними присходит через ядро.

Таким образом подсистемы будут иметь минимальное сцепление, что позволит в дальнейшем легко изменять одну подсистему не затрагивая остальные.

Серверную часть так же можно представить в виде ядра и подсистем, выполняющих определенные действия. Ядро, как и в клиентской части, служит для управления подсистемами и передачи сообщений между ними, а так же между серверной и клиентской частями.

В серверной части имеется практически идентичный набор подсистем, обеспечивающий функциональностью аналогичные подсистемы в клиентской части. Исключение составляют подсистема логирования и подсистема работы с базами данных.

Подсистема логирования обеспечивает администраторов серверной части информацией о работе сервера с той или иной степенью детализации, а подсистема работы с базами данных обеспечивает другие подсистемы необходимой информацией.



### 2.2. Моделирование управления

В качестве модели управления была выбрана модель централизованного управления. Центром управления в данном случае является ядро, которое является менеджером. При возникновении события в системе (получения сообщения) менеджер определяет подсистему, которая должна обработать запрос и отправляет ей данные. В данном случае передачу управления подсистеме можно представить следующим образом:



### 2.2. Декомпозиция на модули.

Каждая подсистема, представленная на структурных диаграммах выше, не зависит от реализации других подсистем и напрямую взаимодействует только с ядром через специализированный интерфейс.

Если одна подсистема хочет получить данные, с которыми работает другая подсистема, то она должна отправить запрос об этих данных ядру, указав в качестве обработчика сообщения желаемую подсистему.

Такая реализация позволяет выделить каждую подсистему в отдельный модуль.

Типы сообщений, обрабатываемые каждой подсистемой, специализированы на круге задач, решаемых подсистемой. Так, например, подсистема управления проектом обрабатывает только сообщения, связанные с созданием, удалением проекта, добавлением файлов, папок, участников и ничего не знает о существовании других подсистем.

В данном случае тип связности всех модулей подсистем – функциональный, так как в целом, каждый модуль решает одну, узкоспециализированную подзадачу. Степень связности – высокая.

Так как подсистема может даже не знать, имеются ли другие подсистемы в данный момент, а обладает лишь информацией о том, какие сообщения можно передавать ядру, то сцепление модулей низкое.

Сцепление модулей подсистем с ядром так же низкое, так как ядро не имеет информации об устройстве каждой подсистемы и не работает с ними напрямую. Все подсистемы работают в ядре через специализированный интерфейс подсистемы.

Для решения подзадач, определенных ядром подсистеме, внутри модуля подсистемы можно так же выделить другие модули.

Интерфейс внутренних модулей подсистемы не специализирован в ядре, а специализирован в самой подсистеме. Поэтому внутренние модули у разных подсистем могут различаться. Но сама подсистема работает со своими внутренними модулями через специализированный интерфейс модуля, благодаря чему сцепление внутренних модулей так же низкое.

Так как внутренние модули специфично решают задачи, определенные для подсистемы в ядре, то их тип связности будет функциональным, а степень связности – низкой.

## 3. Детальное проектирование

### 3.1. Алгоритмы функционирования сервиса.

Диаграмму вариантов использования сервиса со стороны клиента можно представить следующим способом:



Клиент может только подключиться, отключиться и передать серверу сообщение. Весь функционал сервера реализуется обработкой множества различных видов сообщений.

Общий алгоритм функционирования сервера выглядит следующим образом:



Алгоритм работы сервера извне достаточно прост – сервер только принимает сообщения от клиента, производит их обработку и отправляет результат клиенту.

Алгоритм работы клиента так же прост – программа следит за действиями пользователями и при необходимых условиях (выбор определенного пункта меню, нажатие кнопки и т.д.) формирует соответствующие сообщения, которые отправляет на сервер. При получении ответа от сервера программа обрабатывает его и продолжает следить за действиями пользователя.

Алгоритм работы клиентской части можно представить следующим образом:



Получив сообщение сервис должен его обработать. Так как сервис знает о существовании ядра, которое владеет информацией о подсистемах, то при получении сообщения сервис передает его ядру для обработки.

Получив сообщение, ядро находит подсистему, которая должна его обработать, на основании информации, представленной в сообщении.

Если ядро находит подсистему, которая может обработать сообщение, то он передает ей управление. Если же подсистема не найдена, то ядро сообщает о том, что данное сообщение обрабатывать некому.

Непосредственно алгоритм обработки входящего сообщения на стороне сервера можно представить следующим образом:



### 3.2. Модель базы данных

Вся информация, с которой работает сервис, по большому счету хранится в базе данных. Только файлы проекта размещаются на жестком диске, хотя информация о них все равно присутствует в базе.

Модель базы данных можно представить следующим образом:

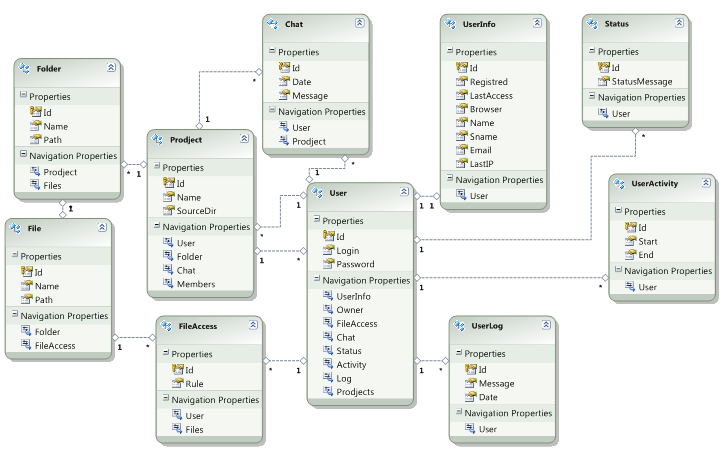


Таблица User содержит информацию о пользователе, необходимую для обеспечения доступа к сервису. Она имеет связи со многими таблицами, в которых находится дополнительная пользовательская информация.

Таблица UserInfo содержит информацию о пользователе – в ней присутствуют поля для заполнения имени, фамилии, браузере пользователя, о дате его регистрации, последнем доступе в систему.

Таблица UserLog хранит лог действий пользователя – время и сообщение.

Таблица UserActivity содержит информацию о том, когда пользователь использовал систему – с какого и по какой период.

Таблица Status содержит пользовательские статусы.

Все вышеперечисленные таблицы необходимы для сбора информации о том, что, когда и сколько времени делал пользователь в системе, для формирования отчетов о проделанной работе.

Еще одна группа таблиц связана с организацией проекта

Таблица Project - представляет проект. Она содержит поля для определения имени и директории, в которой будут храниться исходные коды проекта, а так же содержит ссылки на владельца и участников проекта.

Таблица File содержит список всех файлов проекта. В ней присутствуют поля для определения имени файла и его реального местоположения на диске.

Таблица Folder содержит список всех папок проекта. В ней, так же как в таблице File присутствуют поля для определения имени папки и её реального местоположения на диске.

Таблица FileAccess содержит правила доступа к файлам, определяемые для каждого пользователя. Она необходима для разграничения прав на доступ к проекту, изоляцию ролей участников проекта.

Таблица Chat хранит историю чата проекта. В ней присутствуют поля для определения даты отправки сообщения, отправителя и, собственно, самого сообщения.

Работа с базой данных осуществляется подсистемой работы с базой данных.

В качестве технологии доступа к данным выбрана технология объектно-реляционного отображения, на основе платформы Mindscape Lightspeed, что позволяет отказаться от использования SQL-запросов в программном средстве и в дальнейшем переводить приложение на использование другого сервера баз данных без изменения кода программы.

### 3.3. Диаграммы и описание модулей программы

В основе, как сервисного приложения, так и клиентского, лежит ядро, управляющее запуском и остановом подсистем, передачей им сообщений и слежением за их состоянием.



Класс ядра удобно представить в виде реализации паттерна проектирования singleton, что позволит иметь всего один экземпляр ядра, видимый из любой части программы.

Внутри ядро содержит только список известных подсистем и указатель на то, в какой части программы (серверной или клиентской) это ядро используется, что бы различать, какие сообщения данное ядро должно обрабатывать.

Так же, для управления подсистемами ядро содержит методы запуска, останова и перезагрузки подсистем. Данные методы используются при возникновении непредвиденных ситуаций при работе подсистем.

Все классы исключений, обрабатываемых ядром, выстроены в иерархию.

Во внешнюю среду ядро предоставляет методы для регистрации и отключения подсистем, а так же для приема сообщений.

Структура сообщения, передаваемая как между клиентской и серверной частью, так и между ядром и его подсистемами выглядит следующим образом:



В сервисном сообщении содержится следующая информация:

1. Ядро, которому предназначается сообщение – Handler
2. От какой подсистемы пришло это сообщение – From
3. Какой подсистеме оно предназначается – To
4. Тип сообщения – Type
5. Сообщение, представленное массивом объектов – Message

Типы сообщений, обрабатываемые различными подсистемами, различаются, поэтому в сообщении содержится супер тип перечисления.

Так как, с точки зрения ядра, все подсистемы выглядят одинаково и различаются только множеством обрабатываемых сообщений, то удобно было бы определить интерфейс подсистемы и его реализацию, включающую реализацию общих алгоритмов функционирования всех подсистем:



Интерфейс ISubsystem представляет внешний интерфейс общения подсистемы с ядром, и содержит описание методов запуска, останова, перезагрузки подсистемы, а так же получения информации о работе подсистемы и её состояния.

Реализация интерфейса, а так же обобщенный класс для всех подсистем – Subsystem, является абстрактным и содержит определение только общих методов для всех подсистем, а именно:

1. GetInfo – метод получения информации о подсистеме
2. GetState – метод получения состояния подсистемы
3. Type – метод получения типа подсистемы

Информация о подсистеме упаковывается в специальный класс, содержащий информацию о количестве обработанных сообщений, состоянии и типе подсистемы. Данная информация может быть полезна в случае анализа работы программы, с целью выявления, например, наиболее часто используемых подсистем для их дальнейшей оптимизации.

Все подсистемы, за исключением подсистемы логирования и конфигураций расширяют абстрактный класс Subsystem.

Так как подсистема логирования не должна дополнительно нагружать всю программу, то более целесообразно её представить в качестве реализации паттерна singleton. Это позволит более эффективно реализовать логирование, так как для добавления сообщения в лог-файл не потребуется направлять сообщение ядру, которое бы потом передало его подсистеме, а сразу вызвать метод.



В подсистеме логирования имеется только один метод, Write, который принимает на вход сообщение и его уровень. В зависимости от уровня, подсистема определяет, стоит ли добавлять сообщение в файл, учитывая текущие настройки программы.

Так же имеется указатель на подписчика события добавления сообщения в лог-файл – WriteLog и его события – OnLog. Это позволит сторонним компонентам программы отслеживать запись сообщений, например, для оповещения системного администратора, или просто вывода на экран.

Подсистему конфигурирования, по тем же соображениям, что и подсистему логирования, удобно представить в виде singleton’а:



Подсистема содержит 2 метода:

1. Read – читает указанный параметр из определенной секции файла конфигурации
2. Write – записывает указанный параметр в определенную секцию

Подсистему управления доступом клиента к сервису можно представить следующим образом:



Сама подсистема расширяет класс Subsystem и переопределяет его абстрактные методы – такие так старт, стоп, перезагрузка.

Для переопределения метода отправки сообщения используется модуль, реализующий методы обработки сообщений, посылаемых данной подсистеме.

Перечисление AccessMessages определяет, какие типы сообщений обрабатывает данная подсистема, а именно:

1. Вход пользователя в систему – Login
2. Проверка доступности имени пользователя – CheckLogin
3. Запрос на регистрацию пользователя – Register
4. Получение информации о пользователе – GetInfo

Основываясь на перечислении обрабатываемых сообщений можно представить интерфейс модуля данной подсистемы, реализующий указанный функционал – IAccessModule.

Сам модуль AccessModule выполнен как реализация интерфейса IAccessModule, что позволит в дальнейшем создавать пользовательские модули, обладающие другим функционалом для обработки данных сообщений.

Методы модуля служат для обработки конкретных типов сообщений, название метода соответствует типу обрабатываемого сообщения.

Подсистема управления чатом аналогична по своей архитектуре подсистеме управления доступом:



Сама подсистема так же расширяет класс Subsystem.

Отличительной особенностью является модуль данной подсистемы и обрабатываемые ею типы сообщений.

В перечислении ChatMessages определены следующие типы сообщений:

1. GetMessages – сообщение получения последний сообщений
2. GetHistory – запрос на получение всей истории чата
3. Send – отправка сообщения пользователем
4. Receive – получение сообщения пользователем

Сам модуль так же реализует интерфейс IChatModule, в котором определены методы для обработки сообщений с соответствующими названиями.

Класс ChatMessage определяет сообщение, обрабатываемые чатом. Он содержит информацию об отправителе сообщения, времени его отправки и само сообщение.

Подсистема управления сетью – предназначена для обработки входящих / исходящих сообщений ядра. Она предназначена для шифрования / дешифрования передаваемых сообщений между клиентской и серверной частями.



Архитектура данной подсистемы так же аналогична выше приведенным подсистемам.

Подсистема управления сетью обрабатывает два типа сообщений:

1. Encode – Шифрование входящего сообщения
2. Decode – Дешифрование сообщения

Модуль данной подсистемы реализует обработку этих двух типов сообщений.

Подсистема управления проектами пользователей – ProjectSubsystem – предназначена для обеспечения функционала по управлению проектом.

Она обрабатывает различные запросы, связанные с организацией проектов пользователей.



Сама подсистема так же расширяет абстрактный класс Subsystem.

Перечисление ProjectMessages определяет множество сообщений, обрабатываемых данной подсистемой:

1. CreateProject – запрос на создание проекта
2. DeleteProject – запрос на удаление проекта
3. RunProject – запрос на запуск проекта
4. AddFolder – запрос на добавление папки в проект
5. AddFile – запрос на добавление файла в проект
6. AddMember – запрос на добавление участника в проект
7. RemoveFolder – запрос на удаление папки проекта
8. RemoveFile – запрос на удаление файла проекта
9. RemoveMember – запрос на удаление участника проекта

Модуль ProjectModule, реализующий интерфейс IProjectModule, предоставляет функционал для обработки указанных типов сообщений.

Подсистема управления отчетами – ReportSubsystem – позволяет создавать отчеты о проделанной над проектом работе.



В перечислении ReportMessages определено два типа отчетов:

1. UserReport – отчет о занятости пользователя
2. Projectreport – отчет о проделанной над проектом работе

Данные отчеты собираются на основе информации о статусах участников проекта, датах создания файлов/папок проекта, редактировании отдельных файлов.

Модуль ReportModule предоставляет методы для формирования этих типов отчетов.

Подсистема контроля версий предоставляет управления версиями отдельных файлов проекта.



В перечислении VcsMessages определеные следующие типы обрабатываемых сообщений:

1. CreateRepository – запрос создания репозитория
2. DeleteRepository – запрос удаления репозитория
3. Commit – запрос на создание новой версии файла
4. GetRevisionList – получение списка всей версий файла
5. ViewRevision – получение определенной версии файла

Класс, предоставляющий информацию о версиях файлов, содержит поля, определяющие сам файл и номер версии.

Подсистема, отвечающая за функциональность редактора кода – подсистема управления тестовым редактором – WordProcessorSubsystem.



Данная подсистема обрабатывает три типа сообщений:

1. TypeWord - пользователь ввел слово
2. TepeText – пользователь вводит текст
3. LoadFile – пользователь пытается открыть файл

Сообщения о вводе текста и вводе слова с первого взгляда похожи, но на самом деле они различаются.

В данной курсовой работе, например, не предусмотрена реализация авто дополнения кода, но, переопределив реализацию модуля подсистемы управления текстовым редактором, можно его реализовать, изменив функционал обработки сообщения TypeText.

По сути, сообщения типа TypeText инициализируют начало ввода текста в редакторе кода, а сообщения типа TypeWord – окончание.

Внутри подсистемы, для обработки сообщений, определены две структуры, содержащие данные об изменениях текста:

TextPosition – определяет позицию внесенных изменений. Данная структура содержит номер строки, в которой произошли изменения и позицию, в которой эти изменения были произведены.

Структура TextMessage содержит следующие поля:

1. From –с какой позиции было начало изменение
2. To – до какой позиции
3. OldValue – старое значение текста в этом месте
4. NewValue – введенный текст

Для обработки сообщений отправляемых данной подсистеме, в ней содержится модуль TextModule, который предоставляет необходимые методы.

В целом, архитектура и основные принципы работы всех подсистем, за исключение подсистемы логирования и подсистемы управления конфигурацией – одинаковы.

Подсистемы клиентской части имеют аналогичные архитектуру и принципы построения, что и подсистемы серверной части. Различия заключаются только в функционировании модулей:

1. Модули серверной части служат для обработки запросов, а модули клиентской части – для формирования запросов и отображения результатов в графическом интерфейсе пользователя.
2. Модули клиентской части помимо реализации механизма приема / отправки сообщений реализуют графический интерфейс пользователя.

Исключения, генерируемы подсистемами, так же выделены в иерархию:

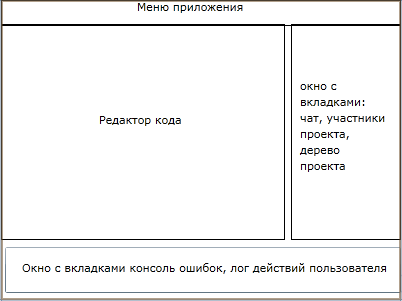


### 3.4. Проектирование интерфейса пользователя

Так как у данной системы существуют аналоги, разрабатываемые крупными корпорациями и сообществами, при проектировании интерфейса стоит учесть интерфейсы аналогичных систем.

Как было описано в предыдущем разделе, интерфейс пользователя реализуют модули подсистем.

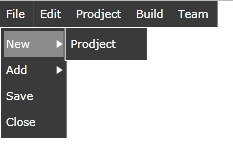
Учитывая интерфейсы аналогичных программ можно представить общий интерфейс системы следующим образом:



Рассмотрим подробнее каждую часть:

Меню приложения: меню аналогичных систем очень похожи.

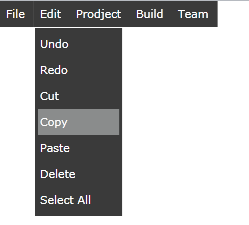
Наибольшую комфортной пользователю можно обеспечить, спроектировав меню аналогичным образом:



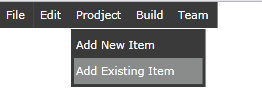
В меню можно выделить 5 категорий, отвечающих за определенные действия.

File – отвечает за создание проекта, сохранение файлов и т.п.

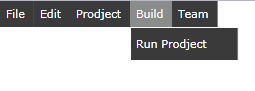
Edit – за изменения в текущем файле:



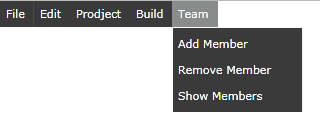
Project – за управление проектом – добавление новых или уже существующих элементов:



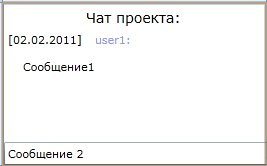
Build – за построение проекта:



Team – за действия с командой, участвующей в проекте:

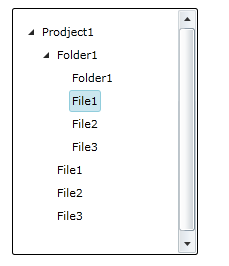


Чат проекта можно представить следующим образом:



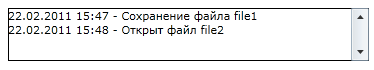
Внизу расположено поле для ввода сообщения текущим пользователем, сверху отображаются входящие сообщения.

Элемент управления, отвечающий за структуру проекта можно представить следующим образом:

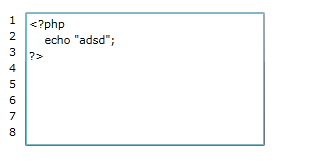


Самый первый элемент обозначает проект. При нажатии на него выпадает список файлов и папок проекта. При выборе файла он открывается в редакторе кода, а при выборе папки – она раскрывается и показывается её содержимое.

Элемент управления отображением информации о действиях пользователя можно представить следующим образом:



Элемент управления, отвечающий за редактор кода, внешне представляет собой просто поле для ввода текста:



Так как данная программа ориентирована на разработчиков программного обеспечения, а так же тот факт, что интерфейс взаимодействия пользователя напоминает аналогичные системы, то у пользователей не должно возникнуть проблем при использовании данной системы.

Высокий уровень подготовки пользователей, а так же простота данной системы позволят без труда осуществить переход с использования аналогичной программы.