МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное образовательное

учреждение высшего профессионального образования

«Ижевский государственный технический университет

имени М.Т. Калашникова»

К защите

Руководитель направления

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

«\_\_\_\_»\_\_\_\_\_\_\_\_\_\_\_20\_\_\_г.

Вылегжанин Сергей Игоревич

Автоматизация процесса сбора и обработки первичной информации

в нефтедобывающей промышленности.

Подсистема хранения данных.

Направление 231000.68 «Программная инженерия»

231000.68-2 – Системы мультимедиа и компьютерная графика

**Диссертация на соискание академической степени**

**магистра техники и технологии**

Магистрант

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Научный руководитель

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Власов В.Г.)

Руководитель программы

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_(Мурынов А.И.)

Ижевск 2014

Реферат

Объем и структура диссертационной работы. Диссертация содержит введение, 3 главы и заключение, изложенные на с. машинописного текста. В работу включены 7 рис., 4 табл., список литературы из 3 наименований.

Система автоматизации предназначена для сбора, обработки и хранения информации с датчиков промышленного оборудования (датчики температуры, давления, клапаны, задвижки и пр.). Подсистема работы с устройствами предназначена для опроса и анализа поведения применяемого оборудования на соответствие нормативным рамкам.

Ключевые слова: OPC-сервер, SCADA, АСУ ТП.

Оглавление

[Реферат 2](#_Toc391306286)

[Оглавление 3](#_Toc391306287)

[Список сокращений 7](#_Toc391306288)

[Введение 8](#_Toc391306289)

[1 Аналитический обзор исследуемой области и формирование требований к системе 10](#_Toc391306290)

[1.1 Аналитический обзор средств автоматизации 10](#_Toc391306291)

[1.1.1 Общая характеристика современных средств автоматизации 10](#_Toc391306292)

[1.1.2 Пример работы оборудования 12](#_Toc391306293)

[1.1.3 Система ZigBee 12](#_Toc391306294)

[1.1.4 Система Круг-2000 13](#_Toc391306295)

[1.2 Основные требования к системе 14](#_Toc391306296)

[1.2.1 Требования к функциональной структуре системы 14](#_Toc391306297)

[1.2.2 Требования к техническому обеспечению. 14](#_Toc391306298)

[1.2.3 Требования к информационному обеспечению. 17](#_Toc391306299)

[1.2.4 Требования к программному обеспечению 17](#_Toc391306300)

[1.2.5 Критерии эффективности функционирования системы 17](#_Toc391306301)

[1.2.6 Перспективность системы, возможности её развития 17](#_Toc391306302)

[1.3 Выводы, постановка цели и задач работы 17](#_Toc391306303)

[2 Разработка архитектуры системы работы с устройствами 19](#_Toc391306304)

[2.1 Разработка архитектуры системы 19](#_Toc391306305)

[2.1.1 Проектирование системы 19](#_Toc391306306)

[2.1.2 Варианты использования системы 24](#_Toc391306307)

[2.1.3 Общая схема работы системы 25](#_Toc391306308)

[2.2 Разработка БД системы 28](#_Toc391306309)

[2.2.1 Развернутая схема БД 28](#_Toc391306310)

[2.2.2 Описание структуры БД 29](#_Toc391306311)

[2.3 Методики работы с проектом 30](#_Toc391306312)

[2.3.1 Система управления версиями 30](#_Toc391306313)

[2.3.2 Системы автоматического тестирования 39](#_Toc391306314)

[2.4 Критерии качества программного обеспечения 42](#_Toc391306315)

[2.5 Полученные результаты и выводы 46](#_Toc391306316)

[3 Разработка подсистем и модулей системы 47](#_Toc391306317)

[3.1 Разработка модуля FacadeBD 47](#_Toc391306318)

[3.1.1 Характеристика задачи 47](#_Toc391306319)

[3.1.2 Входная информация 47](#_Toc391306320)

[3.1.3 Выходная информация 48](#_Toc391306321)

[3.1.4 Функциональное назначение 48](#_Toc391306322)

[3.1.5 Описание алгоритма работы 48](#_Toc391306323)

[3.1.6 Описание реализации 49](#_Toc391306324)

[3.1.7 Результаты испытания 49](#_Toc391306325)

[3.2 Разработка модуля AbstractDatebaseDriver 50](#_Toc391306326)

[3.2.1 Характеристика задачи 50](#_Toc391306327)

[3.2.2 Входная информация 50](#_Toc391306328)

[3.2.3 Выходная информация 50](#_Toc391306329)

[3.2.4 Функциональное назначение 50](#_Toc391306330)

[3.2.5 Описание алгоритма работы 50](#_Toc391306331)

[3.2.6 Описание реализации 51](#_Toc391306332)

[3.2.7 Результаты испытания 51](#_Toc391306333)

[3.3 Разработка модуля MSQLDatebasedriver 51](#_Toc391306334)

[3.3.1 Характеристика задачи 51](#_Toc391306335)

[3.3.2 Входная информация 51](#_Toc391306336)

[3.3.3 Выходная информация 51](#_Toc391306337)

[3.3.4 Функциональное назначение 51](#_Toc391306338)

[3.3.5 Описание алгоритма работы 52](#_Toc391306339)

[3.3.6 Описание реализации 52](#_Toc391306340)

[3.3.7 Результаты испытания 52](#_Toc391306342)

[3.4 Разработка модуля FirebirdeDatebaseDriver 53](#_Toc391306343)

[3.4.1 Характеристика задачи 53](#_Toc391306344)

[3.4.2 Входная информация 53](#_Toc391306345)

[3.4.3 Выходная информация 53](#_Toc391306346)

[3.4.4 Функциональное назначение 53](#_Toc391306347)

[3.4.5 Описание алгоритма работы 54](#_Toc391306348)

[3.4.6 Описание реализации 54](#_Toc391306349)

[3.4.7 Результаты испытания 54](#_Toc391306350)

[Заключение 55](#_Toc391306351)

[Литература 56](#_Toc391306352)

Приложение 1 57

Приложение 2 58

Приложение 3 59

Приложение 4 60

Список сокращений

АСУ ТП — автоматическая система управления техпроцессом.

УСО — устройство связи с объектом.

ПНР — пуско-наладочные работы.

OLE (ObjectLinkingandEmbedding, связывание и внедрение объектов) — технология связывания и внедрения объектов в другие документы и объекты.

OPC (en:OLEforProcessControl) — семейство программных технологий, предоставляющих единый интерфейс для управления объектами автоматизации и технологическими процессами.

SCADA (en:supervisorycontrolanddataacquisition, диспетчерское управление и сбор данных) — программный пакет, предназначенный для разработки или обеспечения работы в реальном времени систем сбора, обработки, отображения и архивирования информации об объекте мониторинга или управления.

Введение

Массовая компьютеризация и информатизация всех отраслей добывающей промышленности в значительной степени повышает производительность и снижает издержки. Однако достигнутые в настоящее время результаты не решают в полной мере проблему автоматизации готовых производств с учетом локальных особенностей. Таким образом, актуальной является задача разработки систем с учетом требований заказчиков.

Целью работы является автоматизация процессов в нефтедобывающей промышленности.

Для достижения поставленной цели решаются следующие задачи:

1) Мониторинг датчиков;

2) Обработка входящих данных;

3) Принятие действий на основе результатов обработки;

4) Сохранение отчетов.

Объектом исследования являются производственные процессы протекающие в нефтедобывающей промышленности, в частности высоконагруженные системы.

Предметом исследования являются методы автоматизации производственных процессов, способы разработки программного обеспечения.

В работе применялись теоретические и экспериментальные методы исследования, такие как нисходящее проектирование и разработка через тестирование.

Теоретические исследования основаны на использовании инженерного подхода к разработке методом нисходящего проектирования, который позволяет минимизировать связность отдельных блоков системы.

В экспериментальных исследованиях разработанных моделей и алгоритмов использовались методы разработки через тестирование, позволяющие автоматизировать проверку качества работы системы.

Достоверность изложенных положений работы подтверждается результатами практического применения разработанных методов, алгоритмов, программных средств на базе ООО «Радиосистемы».

На защиту выносятся результаты разработки системы, а так же исследования методов разработки промышленного программного обеспечения.

Научная новизна полученных результатов определяется изученными подходами к промышленной разработке программного обеспечения.

Практическая ценность работы заключается в снижении затрат на обслуживание нефтедобычи и сопутствующих процессов. Это достигается благодаря повышению качества разработанного ПО.

Разработано программное обеспечение, реализующее задачи мониторинга комплекса устройств, применяемых в нефтедобывающей промышленности.

Разработанный программный комплекс обеспечивает наблюдение за системой в режиме 24/7.

Результаты экспериментальных исследований разработанных средств и методов и оценки их эффективности и возможностей использования при решении различных прикладных задач, а также опыт их внедрения и эксплуатации при создании программных и информационных продуктов в ряде организаций и предприятий подтверждают актуальность системы.

Объем и структура диссертационной работы. Диссертация содержит введение, 3 главы и заключение, изложенные на с. машинописного текста. В работу включены 7 рис., 4 табл., список литературы из 3 наименований.

# Аналитический обзор исследуемой области и формирование требований к системе

В данной главе анализируются существующие системы, выделяются требования и формулируются задачи разрабатываемой системы.

## Аналитический обзор средств автоматизации

### Общая характеристика современных средств автоматизации

Первые попытки автоматизации нефтяных промыслов относятся к 1951 — 1952 гг., когда на нефтяных промыслах «Орджоникидзенефти» (Азербайджанская ССР) была смонтирована система автоматизации и диспетчеризации нефтяных скважин. Однако из-за несовершенства автоматики, сложности аппаратуры дистанционного контроля и управления, а также малой надежности линий связи, испытание системы не дало положительных результатов.

Современные нефтегазодобывающие предприятия представляют собой сложные комплексы технологических объектов, рассредоточенных на больших площадях, размеры которых достигают десятков и сотен квадратных километров. Технологические объекты (скважины, групповые измерительные установки, сепарационные установки, сборные пункты, установки комплексной подготовки нефти и газа, резервуарные парки) связаны между собой через продуктивный пласт и поток продукции, циркулирующей по технологическим коммуникациям. Добыча нефти и газа производится круглосуточно, в любую погоду, поэтому для нормального функционирования нефтегазодобывающего предприятия необходимо обеспечить надежную работу автоматизированного оборудования, а так же дистанционный контроль над работой технологических объектов и их состоянием.

Надежность работы во многом зависит от качества программного обеспечения, которое обрабатывает данные с подконтрольных устройств. Добиться повышения качества позволяют различные методики разработки.

В современной промышленности с целью автоматизации используются системы управления, основанные на различных видах микропроцессоров. Преимуществом данного подхода является простота внесения необходимых изменений, заключающаяся в изменении программного кода обрабатывающей системы без значительных корректировок в аппаратной составляющей. Такой микроконтроллер способен заменить собой несколько десятков аналоговых приборов и регуляторов.

Для упрощения взаимодействия между множеством разнородных устройств, нередко разных производителей, было разработано несколько протоколов передачи информации. Одним из наиболее распространенных является протокол ModBus. Он предоставляет возможность работы как через сеть (локальную/глобальную) используя подвид ModBusTCP, либо же работу через параллельные порты (RS-485, RS-422, RS-232 (последний используется на некоторых персональных ПК, имея название COM-порт)).

### Пример работы оборудования

На рис. 1 приведена схема работы накопительной емкости с клапаном и датчиками уровня.

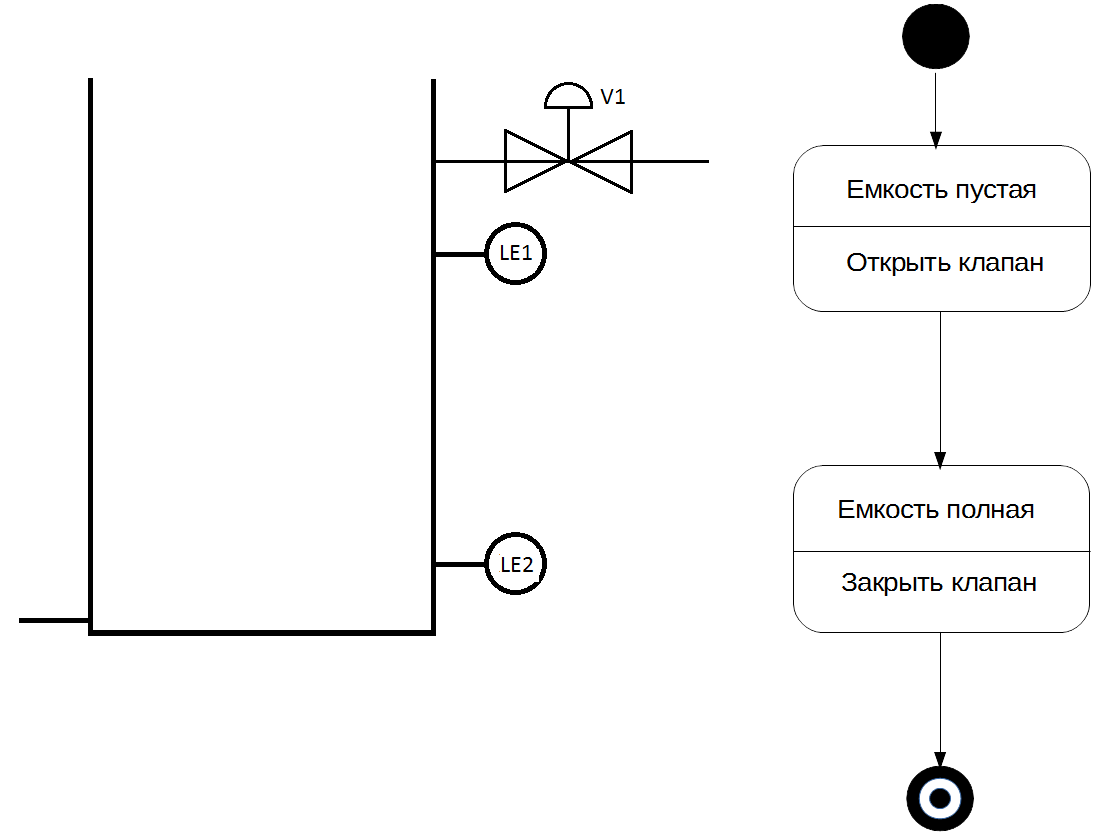


Рис. 1. Схема работы емкости

На схеме:

V1 – клапан, регулирующий поступление жидкости;

LE1 – верхний датчик уровня;

LE2 – нижний датчик уровня.

### Система ZigBee

Является набором беспроводных датчиков и соответствующего ПО. Производитель заявляет способность к самоорганизации и самовосстановлению. Большинство преимуществ и недостатков исходят из применяемой технологии их взаимодействия.

К преимуществам можно отнести:

1) минимальные требования к окружающей среде/инфраструктуре;

2) автономность каждого элемента;

3) возможность контролировать состояние различного типа механического оборудования - в том числе движущихся и вращающихся деталей;

4) высокая гибкость - возможность добавления новых типов датчиков

5)масштабируемость систем - простота при расширении системы новыми сегментами и включения в состав системы новых функциональных подсистем

6)быстрота развертывания системы.

К недостаткам можно отнести:

1) Низкую, в сравнении с проводными типами подключений, помехоустойчивость;

2) Необходимость поддержания автономного питания (аккумуляторы);

3) Возможность перехвата данных команд.

### Система Круг-2000

Позиционируется производителем как средство построения АСУ ТП в области ответственных применений на платформе MicrosoftWindows.

К преимуществам относятся:

1. Технология разработки сертифицирована по ISO 9001:2008;
2. Программные и аппаратные средства автоматического перезапуска (для станций и контроллеров);
3. Поддержка международных стандартов: FastEthernet, RS-485, RS-232, CAN, DeviceNet, TCP/IP (UDP), MODBUS, OPC DA/HDA, COM, DCOM
4. Модульное построение сред разработки и исполнения серверов, станций и контроллеров

К недостаткам можно отнести:

1. Сложность начального обучения;
2. Ориентация системы на работу с MSSQL.

## Основные требования к системе

### Требования к функциональной структуре системы

1. Мониторинг датчиков;
2. Обработка входящих данных;
3. Принятие действий на основе результатов обработки;
4. Сохранение отчетов.

### Требования к техническому обеспечению.

#### Требования к рабочей станции

1) Пропускная способность сети не менее 5 Мб/с;

2) Не менее 1 ГБ ОЗУ;

3) Процессор с частотой от 1.5 ГГц;

4) Дисковое пространство не менее 32 МБ без учета журналов.

#### Перечень поддерживаемых устройств

Задвижка

Для данного устройства должны задаваться параметры:

а) период открытия/закрытия, секунды;

б) время схода с концевика, секунды.

Данное устройство имеет сигналы управления:

а) Открыть(значения: 0, 1);

б) Закрыть(значения: 0, 1);

в) Стоп(значения: 0, 1).

Данное устройство имеет сигналы состояния:

а) Открыта(значения: 0, 1);

б) Закрыта(значения: 0, 1);

в) Авария(значения: 0, 1);

г) Местное управление(значения: 0, 1);

д) Дистанционное управление(значения: 0, 1);

е) Готова к пуску(значения: 0, 1);

В приложении П.1 представлена схема работы с устройством.

Клапан с дискретной регулировкой положения

Для данного устройства должны задаваться параметры:

а) скорость закрытия/открытия клапана, процент/сек.

Данное устройство обеспечивает входные сигналы:

а) Уменьшить % открытия клапана(значения: 0, 1);

б) Увеличить % открытия клапана(значения: 0, 1);

в) Стоп перемещения клапана(значения: 0, 1).

Данное устройство обеспечивает выходные сигналы:

а) Положение клапана, %(значения: 0-100);

б) Полное закрытие(значения: 0, 1);

в) Полное открытие(значения: 0, 1);

б) Авария(значения: 0, 1);

в) Местное управление(значения: 0, 1);

г) Дистанционное управление(значения: 0, 1);

е) Готов к пуску(значения: 0, 1).

В приложении П.3 представлена схема работы с устройством.

Клапан с аналоговой регулировкой положения

Для данного устройства должны задаваться параметры:

а) скорость закрытия/открытия клапана, процент/сек.

Данное устройство обеспечивает входные сигналы:

а) Управление клапаном, %(значения: 0-100);

Данное устройство обеспечивает выходные сигналы:

а) Положение клапана, %(значения: 0-100);

б) Авария(значения: 0, 1);

в) Местное управление(значения: 0, 1);

г) Дистанционное управление(значения: 0, 1);

е) Готов к пуску(значения: 0, 1).

В приложении П.2 представлена схема работы с устройством.

Датчик аналогового сигнала

Для данного сигнала должны задаваться параметры:

а)сигнал типом "пила";

б) период, секунды;

в) значение минимального уровня;

г) значение максимального уровня.

Датчик дискретного сигнала

Для данного сигнала должны задаваться параметры:

а)сигнал типом "ступенька";

б) период, секунды;

в) скважность, %(значения: 0 - 100);.

### Требования к информационному обеспечению.

Большинство журналы представляют собой сведения табличного вида. В качестве контейнера для таких данных было решено использовать формат csv. Поддержка в экспорт данных есть во множестве офисных пакетов (MSExcel пакета MicrosoftOffice, Calc пакетов LibreOffice и OpenOffice и множество других).

### Требования к программному обеспечению

1) WindowsXPи выше;

2) Офисныйпакет (MSOffice, LibreOfficeит. д.) для работы с отчетами;

3) KEPServerEX v5.

### Критерии эффективности функционирования системы

К основным критериям эффективности работы системы относятся:

1. Скорость реакции на поступление событий;
2. Возможность продолжительной работы без перезапуска (24/7);
3. Низкое потребление машинных ресурсов.

### Перспективность системы, возможности её развития

Высокая модульность и использование в качестве OPC-сервера одного из самых распространенных решений, позволяет продолжать разработку с целью охвата большего числа типов поддерживаемого оборудования.

## Выводы, постановка цели и задач работы

Автоматизация процесса тестирования позволяет сократить затраты на разработку ПО благодаря:

1) Увеличению скорости разработки;

2) Повышению качества

3) Быстрая реакция на неполадки;

4) Сохранение журналов работы для последующего анализа.

# Разработка архитектуры системы работы с устройствами

В данной главе производится построение и описание архитектуры системы и сопутствующих структур данных.

## Разработка архитектуры системы

### Проектирование системы

#### Составление технического задания

Исходное задание выдаётся заказчиком и оформляется в виде технических требований. Перевести эти требования на язык предметной области, сформулировать задачу максимально полно и грамотно, обосновать необходимость её решения, осмыслить и уточнить исходные данные — первый этап работы. Исполнитель выполняет его в тесном контакте с заказчиком.

После уточнения и обоснования целей разработки назначают соответствующие им функции. В процессе поиска наиболее полной и точной формулировки строится цепочка функций (дерево целей) — от первоначально предложенной до окончательно принятой. Этому помогает ответ на вопрос «Зачем это нужно?» (и другие вопросы метода контрольных вопросов). В большинстве случаев за приведенной в требованиях целью стоит необходимость выполнения (последовательно или одновременно) нескольких функций. Цепочка функций строится для каждой из них.

Процесс составления ТЗ можно представить в виде следующей последовательности:

Обобщение и абстрагирование.

Увязываются и обобщаются отдельные фрагменты, чтобы, получить цельное, ясное и лаконичное представление о разрабатываемом объекте с учетом возможных изменений.

Абстрагирование предназначено дать такую формулировку требованиям, чтобы избежать предопределения путей решения задачи (не создавать психологических барьеров). Для получения «сильных» решений рекомендуют проводить усиление системы требований и обострение противоречий путем формулирования ИКР.

Проверка на противоречивость.

При наличии нескольких функций часть их по своему действию может оказаться противоречивыми. Для разрешения противоречий эффективно применение эвристических методов. При этом устранение противоречий возможно как на этапе составления ТЗ (изменение формулировок функций, разнесение их действия во времени или в пространстве и т. д.), так и на последующих этапах проектирования.

Условия и ограничения также следует проверять на противоречивость. Так, ограничения могут задавать пустое множество. Подобные противоречия не всегда очевидны: сведения по верхней и нижней границам могут поступать в разное время или помещаться в разных местах ТЗ, быть представлены в неявном виде.

Разграничение требований на условия, ограничения и показатели качества.

Представление требований в виде показателей позволяет получить решения с высокими характеристиками, но такая задача решается сложнее. В качестве показателей выбирают те, которые характеризуют наиболее важные свойства (с целью возможности получения наилучших значений). Для вводимых условий необходимо оценить величину разброса и необходимость указания предельных значений, то есть представления их в виде ограничений.

Параметризация.

Точность суждения и верность выбора зависят от степени конкретности исходных требований, представлены ли они в формализованном или неформализованном виде. Для однозначности выводов все требования должны быть переведены в формализованный вид, то есть указаны характеризующие их параметры, причем такие, которые можно измерить, проконтролировать, рассчитать. Это также позволит выделить дублирующие требования (те, которые характеризуются одними и теми же параметрами) и обобщить их (ввести обобщенные параметры с целью сокращения общего их числа, удельные характеристики).

При решении задачи оптимального проектирования рекомендуют показатели качества приводить к критериальному формализованному виду, то есть назначать им численную меру. Основной метод конкретизации формулировок — построение дерева целей (И или ИЛИ-деревья): исходный показатель декомпозируется до выявления элементарных понятий, однозначно характеризуемых наборами параметров.

Усечение списка требований.

Большой объем информации хотя и способен дать максимально полное представление о решаемой задаче, но оперировать таким объемом сложнее. Для сокращения сведений до разумного объема (под способности каждого конкретного разработчика, соответствие его организационно-техническим, временным ресурсам) можно воспользоваться их ранжированием или разделением на группы обязательных к учету, желательных и несущественных. К обязательным относятся требования следствие реализации, которых существенным образом влияет на выбор вариантов решений. Это — функциональные параметры, условия взаимосвязи систем и их частей и другие. Желательные требования позволяют различить варианты по степени качества.

Заключительным этапом является сведение требований в единый документ и утверждение его заказчиком.

#### Объектно-ориентированная декомпозиция

На этапе проектирования в первую очередь выделяются крупные слабосвязанные блоки (классы анализа) на которые можно разбить основную систему.

Классы анализа наиболее близки к реальным объектам предметной области. Под слабой связанностью подразумевается малое число интерфейсов между этими блоками. После определения интерфейсов взаимодействия происходит разбиение классов анализа на классы проектирования — более специализированные классы, которые инкапсулируют какую-либо функцию/тип данных класса анализа.

Данная схема позволяет производить последующие изменения структуры системы с минимальными трудозатратами.

Предварительная схема системы представлена на рис. 2.

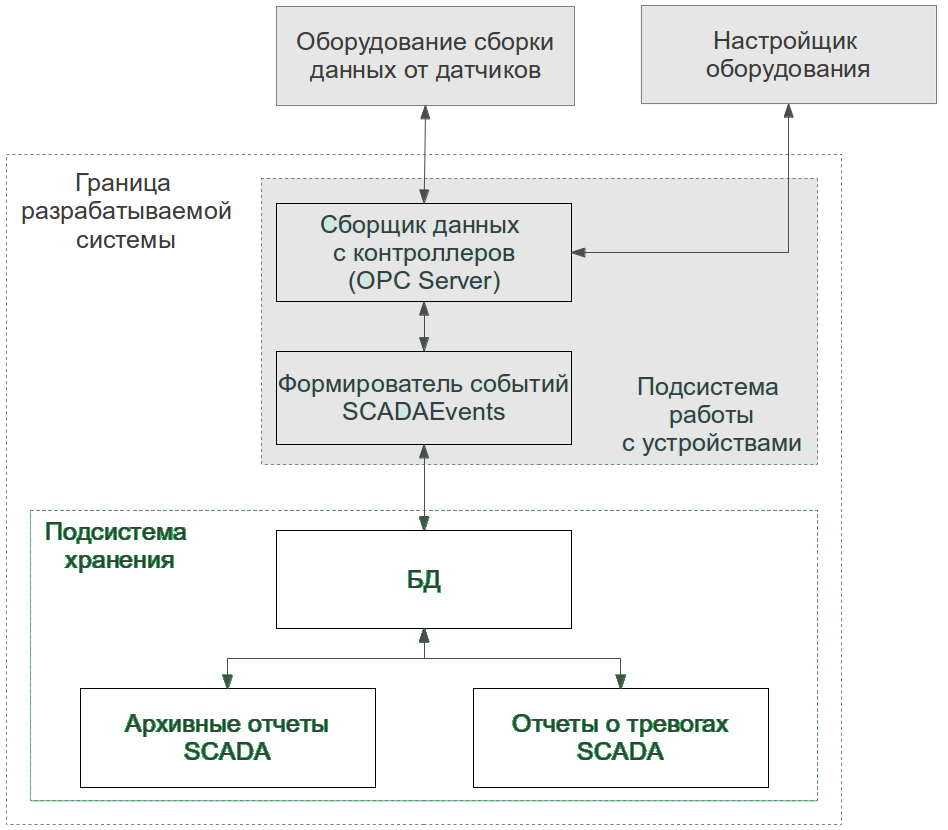


Рис. 2. Предварительная схема системы.

#### Разработка тестов

После выделения классов анализа и проектирования определяются допустимые входные и выходные данные, реакция на получение данных не соответствующего формата и т. д. На основании этих сведений и интерфейсов классов анализа и проектирования строятся тесты на соответствие реализации интерфейсов этим требованиям.

Если в течение разработки требования к системе меняются — то это так же ведет к доработке созданных тестов по новым требованиям.

#### Реализация и тестирование

В процессе написания классов анализа и проектирования производится выполнение составленных тестов для полученных реализаций интерфейсов. В первую очередь отлаживаются классы проектирования в виду их меньшей алгоритмической сложности. Слабая связность и предопределенные интерфейсы позволяют легко применять изменения к низкоуровневым классам.

Если в процессе тестирования какое-либо изменение повлекло за собой провал автоматических тестов (будь-то тесты изменяемого класса или же вышестоящего по иерархии), то производится его анализ, а так же, при необходимости, анализ самих тестов.

### Варианты использования системы

Основное применение системы – промышленное оборудование нефте- и газодобывающей промышленности. Так же возможна адаптация к гидросистемам. Главным фактором в применении является поддержка выбранного оборудования системой KEPServerEX.

### Общая схема работы системы

Схема работы системы представлена на рис. 3.

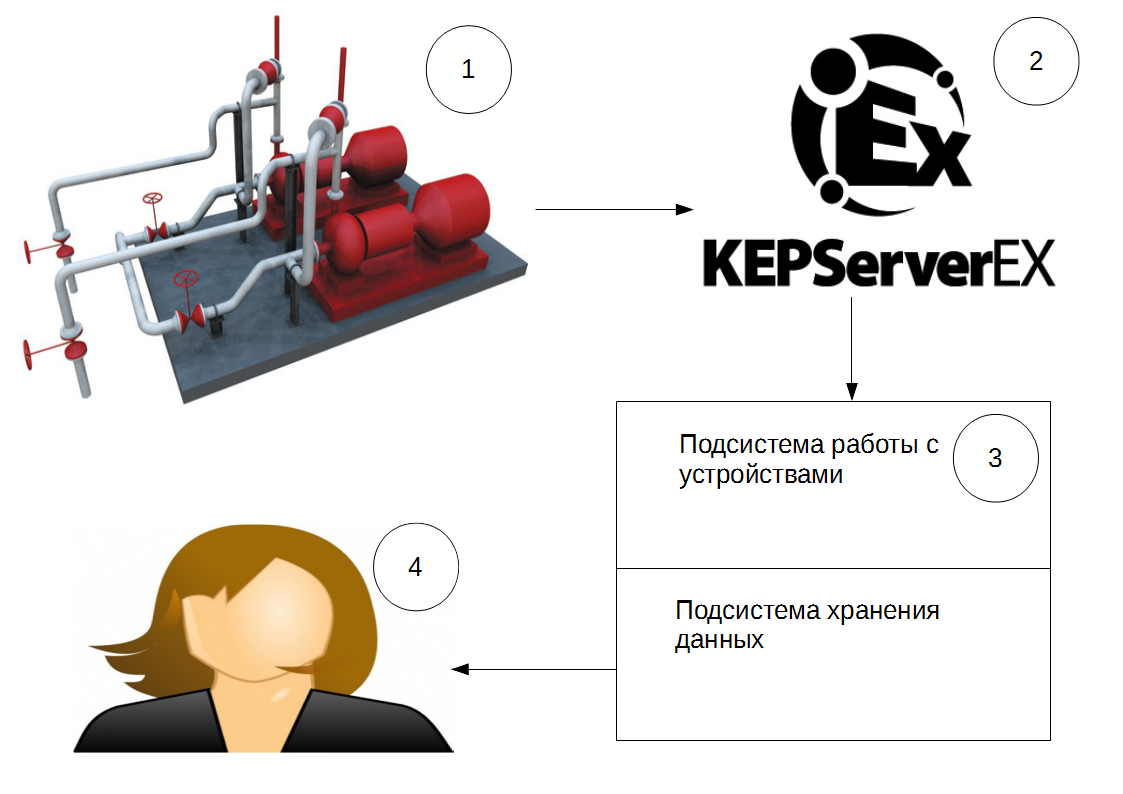


Рис. 3. Схема работы системы.

OPC-сервер KEPServerEXвзаимодействует с различными промышленными устройствами. Полученная информация обрабатывается подсистемой работы с устройствами, передается в систему хранения, которая в последствие предоставляет доступ к данным посредством журналов.

Схема системы хранения данных представлена на рис. 4.

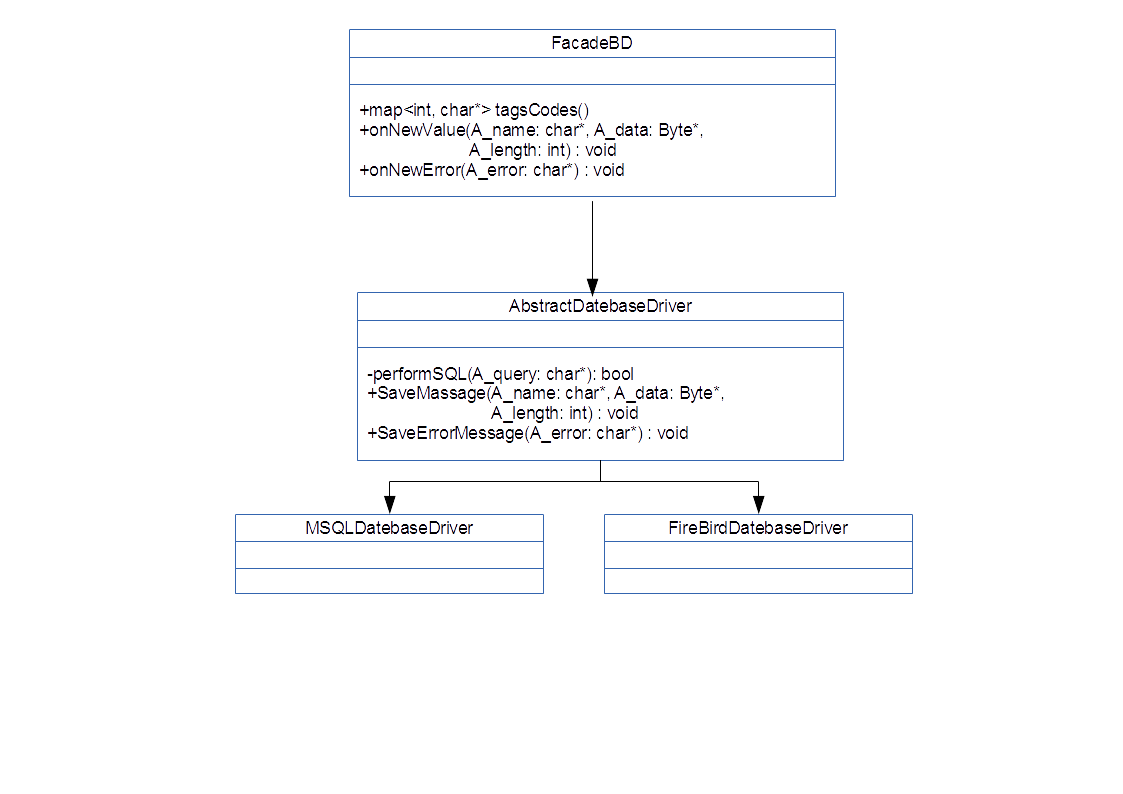


Рис. 4. Схема работы подсистемы хранения данных.

Данные поступают от обработчика тегов и обработчика SCADA-сообщений в модуль FacadeBD. Далее модуль передает сообщения на конкретный драйвер СУБД . Конкретный драйвер реализует диалект соответствующей СУБД. Данный подход позволяет упросить внедрение новых СУБД. Использование разных СУБД дает такое преимущества, как гибкость системы. Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц типа MyISAM, поддерживающие полнотекстовый поиск, так как таблицы InnoDB поддерживают транзакции на уровне отдельных записей. Благодаря открытой архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

В качестве преимущества Firebird можно отметить многоверсионную архитектуру (параллельная обработка оперативных и аналитических запросов: читающие пользователи не блокируют пишущих), компактность (дистрибутив 5mb), высокую эффективность и мощную языковую поддержку для хранимых процедур и триггеров.

## Разработка БД системы

### Развернутая схема БД

Схема БД представлена на рис. 5.

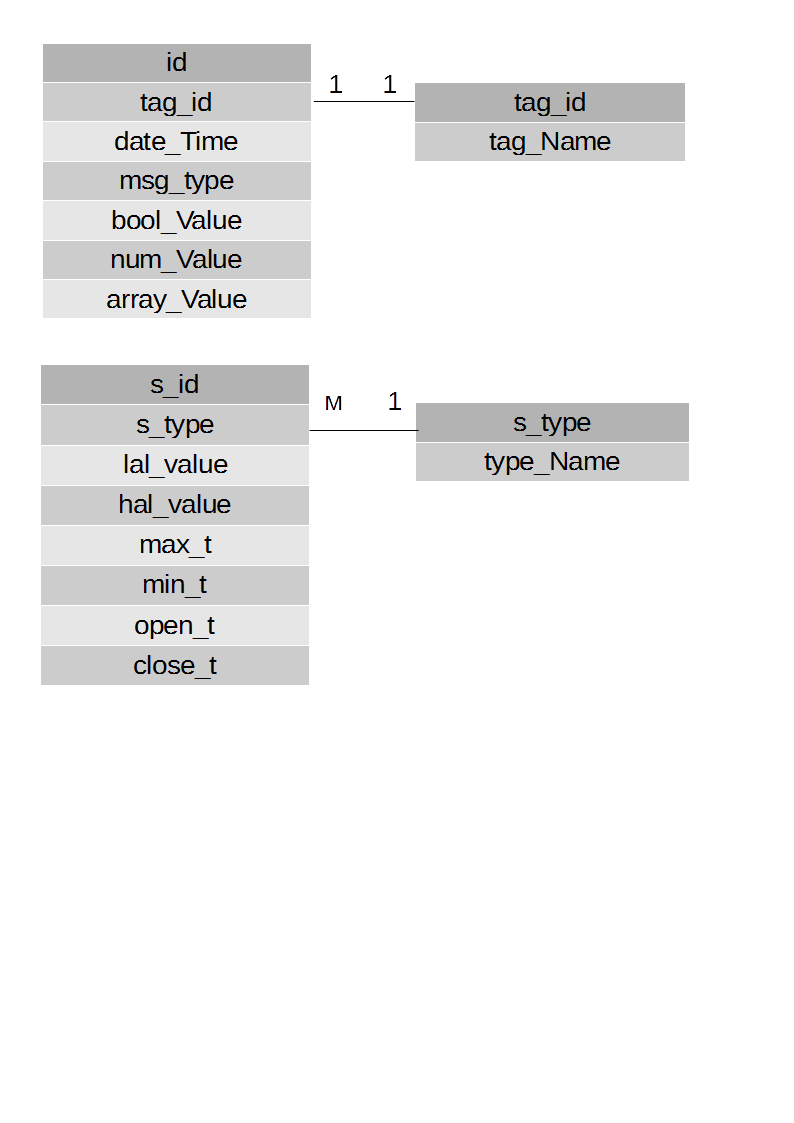


Рис. 5. Схема БД системы

### Описание структуры БД

Основным объектом хранения являются сообщения SCADA, сформированные на основе данных полученных от системы KEPServerEX. Структура сообщения представлена в табл. 1.

Таблица 1

Структура сообщения

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| id | tag\_id | date\_Time | msg\_type | bool\_Value | num\_Value | array\_Value |

id - идентификатор сообщений;

tag\_id - идентификатор тега;

date\_Time - время получения сообщения;

msg\_type - тип сообщения;

bool\_Value - булевое значение;

num\_Value- числовое значение;

array\_Value - массив байт.

Так же в базе хранятся параметры тэгов. Структура представлена в табл. 2.

Таблица 2

Структура таблицы тэгов

|  |  |
| --- | --- |
| tag\_id | tag\_Name |

tag\_id - идентификатор тега;

tag\_Name - имя тега.

Кроме того, в базе хранятся настройки допустимых показателей устройств. Структура представлена в табл. 3.

Таблица 3

Структура параметров устройств

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| s\_id | s\_type | lal\_val | hal\_val | max\_t | min\_t | open\_t | close\_t |

s\_id - идентификатор датчика;

s\_type - тип датчика;

lal\_val - нижний аварийный уровень;

hal\_val - верхний аварийный уровень;

max\_t - верхнее пороговое значение;

min\_t - нижнее пороговое значение;

open\_t - период открытия;

close\_t - период закрытия.

## Методики работы с проектом

### Система управления версиями

Система управления версиями (от англ. VersionControlSystem, VCS или RevisionControlSystem) — программное обеспечение для облегчения работы с изменяющейся информацией. Система управления версиями позволяет хранить несколько версий одного и того же документа, при необходимости возвращаться к более ранним версиям, определять, кто и когда сделал то или иное изменение, и многое другое.

Каждая система управления версиями имеет свои специфические особенности в наборе команд, порядке работы пользователей и администрировании. Тем не менее, общий порядок работы для большинства VCS совершенно стереотипен. Здесь предполагается, что проект, каким бы он ни был, уже существует и на сервере размещён его репозиторий, к которому разработчик получает доступ.

#### Начало работы с проектом

Первым действием, которое должен выполнить разработчик, является извлечение рабочей копии проекта или той его части, с которой предстоит работать. Это действие выполняется с помощью стандартной команды извлечения версии (checkout или clone) либо специальной команды, фактически выполняющей то же самое действие. Разработчик задаёт версию, которая должна быть скопирована, по умолчанию обычно копируется последняя (или выбранная администратором в качестве основной) версия.

По команде извлечения устанавливается соединение с сервером, и проект (или его часть — один из каталогов с подкаталогами) в виде дерева каталогов и файлов копируется на компьютер разработчика. Обычной практикой является дублирование рабочей копии: помимо основного каталога с проектом на локальный диск (либо в отдельный, специально выбранный каталог, либо в системные подкаталоги основного дерева проекта) дополнительно записывается ещё одна его копия. Работая с проектом, разработчик изменяет только файлы основной рабочей копии. Вторая локальная копия хранится в качестве эталона, позволяя в любой момент без обращения к серверу определить, какие изменения внесены в конкретный файл или проект в целом и от какой версии была «отпочкована» рабочая копия; как правило, любая попытка ручного изменения этой копии приводит к ошибкам в работе программного обеспечения VCS.

#### Ежедневный цикл работы

При некоторых вариациях, определяемых особенностями системы и деталями принятого бизнес-процесса, обычный цикл работы разработчика в течение рабочего дня выглядит следующим образом.

Обновление рабочей копии

По мере внесения изменений в основную версию проекта рабочая копия на компьютере разработчика стареет: расхождение её с основной версией проекта увеличивается. Это повышает риск возникновения конфликтных изменений (см. ниже). Поэтому удобно поддерживать рабочую копию в состоянии, максимально близком к текущей основной версии. Для этого разработчик выполняет операцию обновления рабочей копии (update) насколько возможно часто (реальная частота обновлений определяется частотой внесения изменений, зависящей от активности разработки и числа разработчиков, а также временем, затрачиваемым на каждое обновление — если оно велико, разработчик вынужден ограничивать частоту обновлений, чтобы не терять время).

Модификация проекта

Разработчик модифицирует проект, изменяя входящие в него файлы в рабочей копии в соответствии с проектным заданием. Эта работа производится локально и не требует обращений к серверу VCS.

Фиксация изменений

Завершив очередной этап работы над заданием, разработчик фиксирует (commit) свои изменения, передавая их на сервер (либо в основную ветвь, если работа над заданием полностью завершена, либо в отдельную ветвь разработки данного задания). VCS может требовать от разработчика перед фиксацией обязательно выполнить обновление рабочей копии. При наличии в системе поддержки отложенных изменений (shelving) изменения могут быть переданы на сервер без фиксации. Если утверждённая политика работы в VCS это позволяет, то фиксация изменений может проводиться не ежедневно, а только по завершении работы над заданием; в этом случае до завершения работы все связанные с заданием изменения сохраняются только в локальной рабочей копии разработчика.

Ветвления

Делать мелкие исправления в проекте можно путём непосредственной правки рабочей копии и последующей фиксации изменений прямо в главной ветви (в стволе) на сервере. Однако при выполнении объёмных работ такой порядок становится неудобным: отсутствие фиксации промежуточных изменений на сервере не позволяет работать над чем-либо в групповом режиме, кроме того, повышается риск потери изменений при локальных авариях и теряется возможность анализа и возврата к предыдущим вариантам кода в пределах данной работы. Поэтому для таких изменений обычной практикой является создание ветвей (branch), то есть «отпочковывание» от ствола в какой-то версии нового варианта проекта или его части, разработка в котором ведётся параллельно с изменениями в основной версии. Ветвь создаётся специальной командой. Рабочая копия ветви может быть создана заново обычным образом (командой извлечения рабочей копии, с указанием адреса или идентификатора ветви), либо путём переключения имеющейся рабочей копии на заданную ветвь.

Базовый рабочий цикл при использовании ветвей остаётся точно таким же, как и в общем случае: разработчик периодически обновляет рабочую копию (если с ветвью работает более одного человека) и фиксирует в ней свою ежедневную работу. Иногда ветвь разработки так и остаётся самостоятельной (когда изменения порождают новый вариант проекта, который далее развивается отдельно от основного), но чаще всего, когда работа, для которой создана ветвь, выполнена, ветвь реинтегрируется в ствол (основную ветвь). Это может делаться командой слияния (обычно merge), либо путём создания патча (patch), содержащего внесённые в ходе разработки ветви изменения и применения этого патча к текущей основной версии проекта.

Слияние версий

Три вида операций, выполняемых в системе управления версиями, могут приводить к необходимости объединения изменений. Это:

* Обновление рабочей копии (изменения, сделанные в основной версии, сливаются с локальными).
* Фиксация изменений (локальные изменения сливаются с изменениями, уже зафиксированными в основной версии).
* Слияние ветвей (изменения, сделанные в одной ветви разработки, сливаются с изменениями, сделанными в другой).

Во всех случаях ситуация принципиально одинакова и имеет следующие характерные черты:

1. Ранее была сделана копия дерева файлов и каталогов репозитория или его части.
2. Впоследствии и в оригинальное дерево, и в копию были независимо внесены некоторые изменения.
3. Требуется объединить изменения в оригинале и копии таким образом, чтобы не нарушить логическую связность проекта и не потерять данные.

Совершенно очевидно, что при невыполнении условия (2) (то есть если изменения были внесены только в оригинал или только в копию) объединение элементарно — достаточно скопировать изменённую часть туда, где изменений не было. В противном случае слияние изменений превращается в нетривиальную задачу, во многих случаях требующую вмешательства разработчика.

В целом механизм автоматического слияния изменений работает, основываясь на следующих принципах:

* Изменения могут состоять в модификации содержимого файла, создании нового файла или каталога, удалении или переименовании ранее существовавшего файла или каталога в проекте.
* Если два изменения относятся к разным и не связанным между собой файлам и/или каталогам, они всегда могут быть объединены автоматически. Их объединение состоит в том, что изменения, сделанные в каждой версии проекта, копируются в объединяемую версию.
* Создание, удаление и переименование файлов в каталогах проекта могут быть объединены автоматически, если только они не конфликтуют между собой. В этом случае изменения, сделанные в каждой версии проекта, копируются в объединяемую версию. Конфликтующими обычно являются:
  + Удаление и изменение одного и того же файла или каталога.
  + Удаление и переименование одного и того же файла или каталога (в случае, если система поддерживает операцию переименования).
  + Создание в разных версиях файла с одним и тем же именем и разным содержимым.
* Изменения в пределах одного текстового файла, сделанные в разных версиях, могут быть объединены, если они находятся в разных местах этого файла и не пересекаются. В этом случае в объединённую версию вносятся все сделанные изменения.
* Изменения в пределах одного файла, если он не является текстовым, всегда являются конфликтующими и не могут быть объединены автоматически.

Во всех случаях базовой версией для слияния является версия, в которой было произведено разделение сливаемых версий. Если это операция фиксации изменений, то базовой версией будет версия последнего обновления перед фиксацией, если обновление — то версия предыдущего обновления, если слияние ветвей — то версия, в которой была создана соответствующая ветвь. Соответственно, сопоставляемыми наборами изменений будут наборы изменений, сделанных от базовой до текущей версии во всех объединяемых вариантах.

Абсолютное большинство современных систем управления версиями ориентировано, в первую очередь, на проекты разработки программного обеспечения, в которых основным видом содержимого файла является текст. Соответственно, механизмы автоматического слияния изменений ориентируются на обработку текстовых файлов, то есть файлов, содержащих текст, состоящий из строк буквенно-цифровых символов, пробелов и табуляций, разделённых символами перевода строки.

При определении допустимости слияния изменений в пределах одного и того же текстового файла работает типовой механизм построчного сравнения текстов (примером его реализации является системная утилита GNU diff), который сравнивает объединяемые версии с базовой и строит список изменений, то есть добавленных, удалённых и заменённых наборов строк. Минимальной единицей данных для этого алгоритма является строка, даже самое малое отличие делает строки различными. С учётом того, что символы-разделители, в большинстве случаев, не несут смысловой нагрузки, механизм слияния может игнорировать эти символы при сравнении строк.

Те найденные наборы изменённых строк, которые не пересекаются между собой, считаются совместимыми и их слияние делается автоматически. Если в сливаемых файлах находятся изменения, затрагивающие одну и ту же строку файла, это приводит к конфликту. Такие файлы могут быть объединены только вручную. Любые файлы, кроме текстовых, с точки зрения VCS являются бинарными и не допускают автоматического слияния.

Конфликты и их разрешение

Ситуация, когда при слиянии нескольких версий сделанные в них изменения пересекаются между собой, называют конфликтом. При конфликте изменений система управления версиями не может автоматически создать объединённый проект и вынуждена обращаться к разработчику. Как уже говорилось выше, конфликты могут возникать на этапах фиксации изменений, обновления или слияния ветвей. Во всех случаях при обнаружении конфликта соответствующая операция прекращается до его разрешения.

Для разрешения конфликта система, в общем случае, предлагает разработчику три варианта конфликтующих файлов: базовый, локальный и серверный. Конфликтующие изменения либо показываются разработчику в специальном программном модуле объединения изменений (в этом случае там демонстрируются сливаемые варианты и динамически изменяющийся в зависимости от команд пользователя объединённый вариант файла), либо просто помечаются специальной разметкой прямо в тексте объединённого файла (тогда разработчик должен сам сформировать желаемый текст в спорных местах и сохранить его).

Конфликты в файловой системе разрешаются проще: там может конфликтовать только удаление файла с одной из прочих операций, а порядок файлов в каталоге не имеет значения, так что разработчику остаётся лишь выбрать, какую операцию нужно сохранить в сливаемой версии.

Разработка ведется в системе контроля версий, с отдельными ветками для готового проекта и разрабатываемых блоков.

Код, включаемый в основную ветку, должен успешно проходить все тесты. Отдельные ветки свойств могут не соответствовать этому правилу. Фиксация изменений снабжается комментариями о сделанных изменениях (что было изменено и с какой целью).

#### Базовые принципы разработки ПО в VCS

Порядок использования системы управления версиями в каждом конкретном случае определяется техническими регламентами и правилами, принятыми в конкретной фирме или организации, разрабатывающей проект. Тем не менее, общие принципы правильного использования VCS немногочисленны и едины для любых разработок и систем управления версиями.

1. Любые рабочие, тестовые или демонстрационные версии проекта собираются только из репозитория системы. «Персональные» сборки, включающие ещё незафиксированные изменения, могут делать только разработчики для целей промежуточного тестирования. Таким образом, гарантируется, что репозиторий содержит всё необходимое для создания рабочей версии проекта.
2. Текущая версия главной ветви всегда корректна. Не допускается фиксация в главной ветви неполных или не прошедших хотя бы предварительное тестирование изменений. В любой момент сборка проекта, проведённая из текущей версии, должна быть успешной.
3. Любое значимое изменение должно оформляться как отдельная ветвь. Промежуточные результаты работы разработчика фиксируются в эту ветвь. После завершения работы над изменением ветвь объединяется со стволом. Исключения допускаются только для мелких изменений, работа над которыми ведётся одним разработчиком в течение не более чем одного рабочего дня.
4. Версии проекта помечаются тегами. Выделенная и помеченная тегом версия в дальнейшем не изменяется.

### Системы автоматического тестирования

#### Выбор системы тестирования

Разработка системы выполняется с использованием языка программирования C++, который в исходном виде не предполагает наличия автоматических средств тестирования.

За время существования языка было создано значительное количество сторонних библиотек для восполнения указанного недостатка. Наиболее распространёнными системами являются:

* CppUnit;
* QtTest;
* CUTE;
* Google Test;
* Parasoft C/C++test.

Для применения в разрабатываемой системе был выбрана библиотекаCppUnit, в виду удобства и наличия опыта работы с ней.

Выбранная библиотека позволяет проводить автоматическое модульное тестирование. Суть модульного тестирования состоит в том, чтобы писать тесты для каждой нетривиальной функции или метода. Это позволяет достаточно быстро проверить, не привело ли очередное изменение кода к регрессии, то есть к появлению ошибок в уже оттестированных местах программы, а также облегчает обнаружение и устранение таких ошибок.

#### Преимущества подхода

Поощрение изменений

Модульное тестирование позже позволяет программистам проводить рефакторинг, будучи уверенными, что модуль по-прежнему работает корректно (регрессионное тестирование). Это поощряет программистов к изменениям кода, поскольку достаточно легко проверить, что код работает и после изменений.

Упрощение интеграции

Модульное тестирование помогает устранить сомнения по поводу отдельных модулей и может быть использовано для подхода к тестированию «снизу вверх»: сначала тестируя отдельные части программы, а затем программу в целом.

Документирование кода

Модульные тесты можно рассматривать как «живой документ» для тестируемого класса. Клиенты, которые не знают, как использовать данный класс, могут использовать юнит-тест в качестве примера. Юнит-тест - процесс в программировании, позволяющий проверить на корректность отдельные модули исходного кода программы.

Отделение интерфейса от реализации

Поскольку некоторые классы могут использовать другие классы, тестирование отдельного класса часто распространяется на связанные с ним. Например, класс пользуется базой данных; в ходе написания теста программист обнаруживает, что тесту приходится взаимодействовать с базой. Это ошибка, поскольку тест не должен выходить за границу класса. В результате разработчик абстрагируется от соединения с базой данных и реализует этот интерфейс, используя свой собственный mock-объект. Это приводит к менее связанному коду и минимизирует зависимости в системе.

#### Ограничения применения модульного тестирования

Сложный код

Тестирование программного обеспечения — комбинаторная задача. Например, каждое возможное значение булевской переменной потребует двух тестов: один на вариант TRUE, другой — на вариант FALSE. В результате на каждую строку исходного кода потребуется 3?5 строк тестового кода.

Как и любая технология тестирования, модульное тестирование не позволяет отловить все ошибки программы. В самом деле, это следует из практической невозможности трассировки всех возможных путей выполнения программы, за исключением простейших случаев.

Результат известен лишь приблизительно

Например, в математическом моделировании. Бизнес-приложения зачастую работают с конечными и счётными множествами, научные — с континуальными. Поэтому сложно подобрать тесты для каждой из ветвей программы, сложно сказать, верен ли результат, выдерживается ли точность, и т. д. А во многих случаях качество моделирования определяется «на глаз», и последний результат записывается как «опорный». Если найдено расхождение, новый результат проверяют вручную и выясняют, какой качественнее: старый или новый.

Ошибки интеграции и производительности

Кроме того, происходит тестирование каждого из модулей по отдельности. Это означает, что ошибки интеграции, системного уровня, функций, исполняемых в нескольких модулях, не будут определены. Кроме того, данная технология бесполезна для проведения тестов на производительность. Таким образом, модульное тестирование более эффективно при использовании в сочетании с другими методиками тестирования.

Документирование процесса разработки

Для получения выгоды от модульного тестирования требуется строго следовать технологии тестирования на всём протяжении процесса разработки программного обеспечения. Необходимо хранить не только записи обо всех проведённых тестах, но и обо всех изменениях исходного кода во всех модулях. С этой целью следует использовать систему контроля версий ПО. Таким образом, если более поздняя версия ПО не проходит тест, который был успешно пройден ранее, будет несложным сверить варианты исходного кода и устранить ошибку. Также необходимо убедиться в неизменном отслеживании и анализе неудачных тестов. Игнорирование этого требования приведёт к лавинообразному увеличению неудачных тестовых результатов.

## Критерии качества программного обеспечения

Ранее в литературе [2] были описаны внешние и внутренние характеристика качества ПО. Под внешними характеристиками подразумеваются те, которые осознает пользователь приложения.

Корректность — отсутствие/наличие дефектов в спецификации, проекте и реализации системы.

Практичность — легкость изучения и использования системы.

Эффективность — степень использования системных ресурсов. Эта характеристика учитывает такие факторы, как быстродействие приложения и требуемый им объем памяти.

Надежность — способность системы выполнять необходимые функции в предопределенных условиях; средний интервал между отказами.

Целостность — способность системы предотвращать неавторизованный или некорректный доступ к своим программам и данным. Идея целостности подразумевает ограничение доступа к системе для неавторизованных пользователей, а также обеспечение правильности доступа к данным, т. е. одновременное изменение взаимосвязанных данных, хранение только допустимых значений и т. д.

Адаптируемость — возможность использования системы без ее изменения в тех областях или средах, на которые она не была ориентирована непосредственно.

Правильность — степень безошибочности системы, особенно в отношении вывода количественных данных. Правильность характеризует выполнение системой ее функций, а не то, создана ли она корректно. Этим правильность отличается от корректности.

Живучесть — способность системы продолжать работу при вводе недопустимых данных или в напряженных условиях.

Внешние характеристики — единственная категория свойств ПО, которая волнует пользователей. Пользователей беспокоит легкость работы сПО, а не легкость его изменения. Их заботит корректность ПО, а не удобочитаемость или структурированность кода. Программистов же волнуют и внешние характеристики ПО, и внутренние.

Удобство сопровождения — легкость изменения программной системы с целью реализации дополнительных возможностей, повышения быстродействия, исправления дефектов и т. д.

Гибкость — возможный масштаб изменения системы с целью использования ее в тех областях или средах, на которые она не была непосредственно ориентирована.

Портируемость — легкость изменения системы с целью использования в средах, на которые она не была ориентирована непосредственно.

Возможность повторного использования — масштабность и легкость использования частей системы в других системах.

Удобочитаемость — легкость чтения и понимания исходного кода системы, особенно на детальном уровне отдельных операторов.

Тестируемость — возможная степень выполнения блочного и системного тестирования программы и проверки ее соответствия требованиям.

Понятность — легкость понимания системы и на уровне общей организации, и на детальном уровне отдельных операторов. Понятность характеризует согласованность системы на более общем уровне, чем удобочитаемость.

Различие между внутренними и внешними характеристиками качества размыто, потому что на некотором уровне первые влияют на вторые. Если программа недостаточно понятна или неудобна в сопровождении, в ней трудно исправлять дефекты, что в свою очередь влияет на такие внешние характеристики, как корректность и надежность. ПО, страдающее от недостатка гибкости, нельзя улучшить в ответ на запросы пользователей, что сказывается на его практичности. Суть сказанного в том, что одни характеристики качества упрощают использование ПО пользователями, а другие написание ПО программистами.

Улучшение некоторых аспектов качества неизбежно приводит к ухудшению других. Необходимость согласования решения с рядом противоречащих целей делает разработку ПО поистине инженерной дисциплиной. Взаимосвязь внешних характеристик качества пояснена в табл. 4. Подобные отношения имеют место и между внутренними характеристиками.

Таблица 4

Взаимосвязь характеристик качества ПО

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Корректность | Практичность | Эффективность | Надежность | Целостность | Адаптируемость | Правильность | Живучесть |
| Корректность | ▲ |  | ▲ | ▲ |  |  | ▲ | ▼ |
| Практичность |  | ▲ |  |  |  | ▲ | ▲ |  |
| Эффективность | ▼ |  | ▲ | ▼ | ▼ | ▼ | ▼ |  |
| Надежность | ▲ |  |  | ▲ | ▲ |  | ▲ | ▼ |
| Целостность |  |  | ▼ | ▲ | ▲ |  |  |  |
| Адаптируемость |  |  |  |  | ▼ | ▲ |  | ▲ |
| Правильность | ▲ |  | ▼ | ▲ |  | ▼ | ▲ | ▼ |
| Живучесть | ▼ | ▲ | ▼ | ▼ | ▼ | ▲ | ▼ | ▲ |

Самый интересный аспект этой таблицы в том, что концентрация на одной конкретной характеристике не всегда предполагает компромисс с другой характеристикой. Одни характеристики связаны между собой обратным отношением, другие — прямым, а третьи вообще не зависят друг от друга. Так, корректность характеризует степень, в которой работа системы соответствует спецификации.

Живучесть характеризует способность системы продолжать работу даже в непредвиденных условиях. Стремление повысить корректность приводит к снижению живучести и наоборот. В то же время концентрация на адаптируемости повышает живучесть и наоборот.

Табл. 4 иллюстрирует только типичные отношения между характеристиками качества. В конкретном проекте две характеристики могут быть связаны и нетипичным отношением. Поэтому при размышлении о качестве полезно думать о конкретных целевых характеристиках и о том, способствует одна из них достижению другой или препятствует.

## Полученные результаты и выводы

В процессе разработки выполнена объектно-ориентированная декомпозиция системы. Выделены классы анализа и проектирования, а так же составлена структура базы данных системы. На основании выполненной работы необходимо реализовать полученные объекты в выбранной среде разработки.

Были выделены следующие блоки:

OPC-клиент –поддержание соединения с OPC-сервером (KEPServerEX);

Обработчик тэгов – анализ полученных от OPC-сервера данных;

Группировщик – выполняет преобразование сигналов в абстрактные устройства;

SCADA-обработчик – анализ поведения устройства и генерация SCADA-сообщений с передачей в систему хранения.

# Разработка подсистем и модулей системы

В данной главе описывается процесс реализации полученной на этапе проектирования архитектуры. Проводились исследование работы модулей на компьютере со следующими характеристиками:

* Inter(R) Core(TM) i5-2300 CPU @ 2.80GHz;
* ОЗУ - 8 ГБ;
* 2Тб дискового пространства.

## Разработка модуля FacadeBD

### Характеристика задачи

В задачи модуля входит:

* сохранение SCADA-сообщения;
* сохранение пары tag:id;
* сохранение установок датчиков;
* загрузка SCADA-сообщения;
* загрузка пары tag:id;
* загрузка установок датчиков.

### Входная информация

На вход поступают теги, параметры настроек датчиков, SCADA-сообщения. Формат данных SCADA-сообщения: [id; tag\_id; date\_Time; msg\_type; bool\_Value; num\_Value; array\_Value], где

* id - идентификатор сообщений;
* tag\_id - идентификатор тега;
* date\_Time - время получения сообщения;
* msg\_type - тип сообщения;
* bool\_Value - булевое значение;
* num\_Value- числовое значение;
* array\_Value - массив байт.

Формат данных тега: [tag\_id, tag\_name], где

* tag\_id - идентификатор тега;
* tag\_Name - имя тега.

Формат данных настроек датчика: [s\_id, s\_type, lal\_val, hal\_val, max\_t, min\_t, open\_t, close\_t], где

* s\_id - идентификатор датчика;
* s\_type - тип датчика;
* lal\_val - нижний аварийный уровень;
* hal\_val - верхний аварийный уровень;
* max\_t - верхнее пороговое значение;
* min\_t - нижнее пороговое значение;
* open\_t - период открытия;
* close\_t - период закрытия.

### Выходная информация

Данные полученные на входе передаются в экземпляр соответствующего СУБД без предварительной обработки.

### Функциональное назначение

Назначением модуля является прием и передача сообщений на конкретный драйвер СУБД . Модуль является универсальным прозрачным интерфейсом для доступа к различным СУБД.

### Описание алгоритма работы

Схема реализации представлена на рис. 4.



Рис. 4. Структура FacaedBD

### Описание реализации

На основе разработанного класса анализа реализуется интерфейс. Интерфейс отвечает за взаимодействие с обработчиком тегов и обработчиком SCADA-сообщений.

### Результаты испытания

По требованиям к системе при обработке потока данных в 250 сообщения/с нагрузка процессора не должно превышать 15%. Результаты испытания представлены в табл. 5.

Таблица 5

Результаты испытания

|  |  |  |
| --- | --- | --- |
| № | Поток данных, сообщения/с | Полученный результат, % |
| 1 | 100 | 8 |
| 2 | 250 | 15 |
| 3 | 500 | 27 |

## Разработка модуля AbstractDatebaseDriver

### Характеристика задачи

Основная задача модуля – организация использования функций объекта, недоступного для модификации, через специально созданный интерфейс.

### Входная информация

Входной информацией являются данные, приходящие от модуля Facade BD.

### Выходная информация

Выходной информацией являются записи в СУБД и журналы. Журналы представляют собой набор SCADA-сообщений, отфильтрованный пользователем.

### Функциональное назначение

Определение интерфейса доступа к СУБД для последующей реализации классов доступа.

### Описание алгоритма работы

Схема работы представлена на рис. 5.



Рис. 5. Схема AbstractDatebaseDriver.

### Описание реализации

AbstractDatebaseDriver разработан в соответствии с шаблоном Адаптер.

Шаблон Адаптер предусматривает создание класса оболочки с требуемым интерфейсом, так же позволяет включать уже существующие объекты в новые объектные структуры, независимо от различий в их интерфейсах.

### Результаты испытания

В результате моделирования было установлено, что модуль выполняет свою функцию.

## Разработка модуля MSQLDatebasedriver

### Характеристика задачи

Модуль предназначен для непосредственного взаимодействия с СУБД MySQL.

### Входная информация

* сохранение SCADA-сообщения;
* сохранение пары tag:id;
* сохранение установок датчиков.

### Выходная информация

Сформированный отчет по MySQL запросу. Загрузка настроек датчиков из СУБД в SCADA обработчик.

### Функциональное назначение

Модуль MSQLDatebasedriver предназначен для решения таких функциональных задач как :

* Установка соединения с сервером MySQL;
* поддержка соединения с сервером MySQL;
* принимать сообщения SCADA;
* сообщения SCADA в СУБД;
* по запросу оператора формировать отчет;
* загрузка SCADA обработчик настроек датчиков;
* реализация SQL запросов с особенностями MySQL диалекта.

### Описание алгоритма работы

В результате работы программы, накапливаются данные. По приходу запроса строятся шаблоны с вставкой соответствующих данных из СУБД.

### Описание реализации

MSQLDatebasedriver разработан в соответствии с шаблоном Заместитель (Proxy). Заместитель - структурный шаблон проектирования, который предоставляет объект, который контролирует доступ к другому объекту, перехватывая все вызовы (выполняет функцию контейнера).

К плюсам данного решения можно отнести:

* удаленный заместитель;
* виртуальный заместитель выполняющий оптимизацию;
* защищающий заместитель;

### К недостаткам можно отнести резкое увеличение времени отклика.

### Результаты испытания

По результатам отображенным в табл. 6 видно, что модуль справляется с возложенными на него задачами.

Таблица 6

Результаты испытания

|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество тегов | Время записи, мс | Время чтения, мс |
| 1 | 1000 | 25 | 2 |
| 2 | 10000 | 240 | 23 |
| 3 | 20000 | 505 | 45 |
| 4 | 30000 | 700 | 67 |
| 5 | 40000 | 945 | 98 |

## Разработка модуля FirebirdeDatebaseDriver

### Характеристика задачи

Модуль предназначен для непосредственного взаимодействия с СУБД Firebirde.

### Входная информация

* сохранение SCADA-сообщения;
* сохранение пары tag:id;
* сохранение установок датчиков.

### Выходная информация

Сформированный отчет по Firebirde запросу. Загрузка настроек датчиков из СУБД в SCADA обработчик.

### Функциональное назначение

Модуль FirebirdeDatebaseDriver предназначен для решения таких функциональных задач как :

* Установка соединения с сервером Firebirde;
* поддержка соединения с сервером Firebirde;
* принимать сообщения SCADA;
* сообщения SCADA в СУБД;
* по запросу оператора формировать отчет;
* загрузка SCADA обработчик настроек датчиков;
* реализация SQL запросов с особенностями Firebirdeдиалекта.

### Описание алгоритма работы

В результате работы программы, накапливаются данные. По приходу запроса строятся шаблоны с вставкой соответствующих данных из СУБД.

### Описание реализации

FirebirdeDatebaseDriver разработан в соответствии с шаблоном Заместитель (Proxy). Особенности описаны в разделе 3.3.6.

### Результаты испытания

По результатам отображенным в табл. 7 видно, что модуль выполняет свою функцию быстрее MySQL.

Таблица 7

Результаты испытания

|  |  |  |  |
| --- | --- | --- | --- |
| № | Количество тегов | Время записи, мс | Время чтения, мс |
| 1 | 1000 | 9 | 2 |
| 2 | 10000 | 80 | 20 |
| 3 | 20000 | 170 | 41 |
| 4 | 30000 | 260 | 63 |
| 5 | 40000 | 330 | 92 |

Заключение

В процессе разработки были изучены и применены такие методики, как стратегия тестирования, целевые характеристики качества ПО. Так же были получены знания в области промышленных автоматизированных систем. Проведена работа над составлением технического задания и документации.

Данные методики повысили качество ПО, уменьшили количество ошибок на этапах проектирования и конструирования.

Результатом проведенной работы стала система автоматизации, нацеленная на работу с одним из самых распространенных OPC-серверов.

Высокая модульность системы позволяет с легкостью адаптировать ее под конкретные задачи и производить дальнейшие улучшения, как то поддержка новых устройств, СУБД и, при некоторых доработках, взаимодействие с различными OPC-серверами.

Литература

1. Горев С.М. Автоматизация производственных процессов нефтяной и газовой промышленности. Курс лекций. Часть 1. — Петропавловск-Камчатский: КамчатГТУ, 2003. – 121 с.
2. Макконнелл С. Совершенный код. Мастер класс – СПб.: Питер, 2005. – 896 с.
3. Требования и рекомендации по содержанию, оформлению, представлению и защите диссертации на соискание академической степени магистра техники и технологии
4. Соболева В.П. Методические указания по оформлению курсовых и

дипломных работ. — Ижевск: Издательство ИжГТУ, 2008.– 25с.

Приложение 1

АЛГОРИТМ УПРАВЛЕНИЯ СИГНАЛАМИ ЗАДВИЖКИ

На рис. П.1 представлена схема управления сигналами задвижки.



Рис. П.1. Схема управления сигналами задвижки.

Приложение 2

АЛГОРИТМ УПРАВЛЕНИЯ СИГНАЛАМИ КЛАПАНА

С АНАЛОГОВОЙ РЕГУЛИРОВКОЙ ПОЛОЖЕНИЯ

На рис. П.2 представлена схема управления сигналами клапана с аналоговой регулировкой положения.



Рис. П.2. Схема управления сигналами клапана с аналоговой регулировкой положения

Приложение 3

АЛГОРИТМ УПРАВЛЕНИЯ СИГНАЛАМИ КЛАПАНА

С ДИСКРЕТНОЙ РЕГУЛИРОВКОЙ ПОЛОЖЕНИЯ

На рис. П.3 представлена схема управления сигналами клапана с дискретной регулировкой положения.



Рис. П.3. Схема управления сигналами клапана с дискретной регулировкой положения

ПРИЛОЖЕНИЕ 4

АЛГОРИТМ РАБОТЫ МОДУЛЯ FACADEBD

На рис. П.4 представлена схема работы модуля FacadeBD.

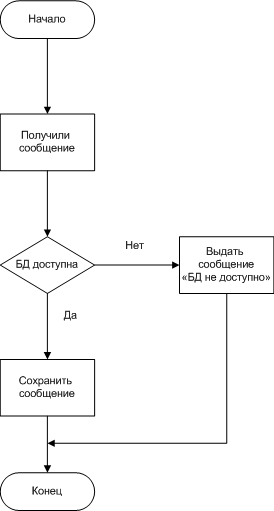


Рис. П.4. представлена схема работы модуля FacadeBD