МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ РФ

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет Вычислительной техники

Кафедра Вычислительная техника

Направление подготовки 09.03.01 «Информатика и вычислительная техника»

Профиль Вычислительные машины, комплексы, системы и сети

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА БАКАЛАВРА

на тему

Клиент-серверное приложение “Планировщик задач”

Студент 3.06.18 Васильев Никита Николаевич
(подпись, дата)

Руководитель Дорошенко И.Н.
(подпись, дата) 3.06.18

Нормоконтролёр 17.06.18 Кучин А.В.
(подпись, дата)

Работа допущена к защите (протокол заседания кафедры от 25.05.2018 № 38-09-12)

Заведующий кафедрой Пащенко Д.В.
(подпись)

Работа защищена с отметкой отл. (протокол заседания ГЭК от 19.06.18. № 3)

Секретарь ГЭК Гурин Е.И.
(подпись)

Пенза, 2018

Федеральное государственное бюджетное образовательное учреждение
высшего образования
ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ПОЛИТЕХНИЧЕСКИЙ ИНСТИТУТ

Факультет	*Кафедра*
Вычислительной техники	Вычислительная техника

«Утверждаю»
Заведующий кафедрой
_____Д.В. Пащенко
«__»_____2018 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ
БАКАЛАВРА

1. Студент Васильев Никита Николаевич гр. 14ВВ1 факультета вычислительной техники.

2.Тема работы: «Клиент-серверное приложение “Планировщик задач”».

Тема утверждена приказом ПГУ № 398а/о от 03.05.2018 г.

3. Руководитель работы к.т.н., доцент каф. ВТ ПГУ Дорошенко И.Н.

4. Задание на работу (назначение разработки, исходные данные и т.п.).
Назначение разработки – клиент-серверное приложение, предназначенное для планирования выполнения задач на ЭВМ. Необходимо предусмотреть возможность подключения к удаленной ЭВМ для управления её задачами.

5. Перечень подлежащих разработке вопросов.
Требуется произвести анализ предметной области, выбрать архитектуру приложения, определить языки программирования для клиентского и серверного приложения, выбрать способ взаимодействия клиента и сервера, разработать протокол взаимодействия.

6. Календарный график выполнения работы

№ п/п	Наименование этапов работы	Объем работы	Срок выполнения	По... руко...
1.	Анализ предметной области	10	14.03.18	Дор...
2.	Выбор архитектуры приложения, языков программирования, способа взаимодействия	10	23.03.18	Дор...
3.	Разработка протокола взаимодействия	10	31.03.18	Дор...
4.	Разработка серверной части	20	09.04.18	Дор...
5.	Разработка клиентской части	20	26.04.18	Дор...
6.	Тестирование и отладка	10	11.05.18	Дор...
7.	Оформление выпускной работы	20	29.05.18	Дор...

Дата выдачи задания "5" марта 2018 г.

Руководитель бакалаврской работы Дорошенко 5.03.18 Дорошенко И...
(подпись, дата)

Задание к исполнению принял студент Васильев 5.03.18 Васильев Н.Н.
(подпись, дата)

Работу к защите допустить
Декан факультета 15.06.2018 Фионова Л.Р.
(подпись, дата)

РЕФЕРАТ

Бакалаврская работа содержит 200 листов, 37 рисунков, 5 приложения.

КЛИЕНТ-СЕРВЕРНОЕ ПРИЛОЖЕНИЕ ПЛАНИРОВЩИК ЗАДАЧ.

Цель работы – разработать клиент-серверное приложение, осуществляющее автономное управление планированием и запуском задач.

Объектом разработки является клиент-серверное приложение, состоящее из программы с графическим интерфейсом и службы работающей в фоновом режиме. Клиент и сервер осуществляют совместную работу по планированию и запуску задач.

В результате проведенной работы реализован проект, удовлетворяющий поставленной задаче, а именно, создано клиент-серверное приложение планирования задач, автономно управляющее запуском задач.

Данное приложение рекомендуется использовать для администрирования операционных систем рабочих станций. Приложение может быть внедрено на предприятиях, где требуется поддерживать работу большого количества рабочих станций. Так же приложение может использоваться частными лицами для управления работой своего компьютера удаленно по сети.

		№ докум.	Подпись	ПГУ 1.090301.02.001 ПЗ	
Разраб.	Васильев Н. Н.			Клиент-серверное приложение "Планировщик задач" Пояснительная записка	Лит. / Лист 4 / Листов 200
Провер.	Дорошенко И.Н.				
Н. Контр	Кучин А.В.				ФВТ, каф. ВТ гр. 14ВВ1
Утверд.	Пащенко Д.В.				

Содержание

РЕФЕРАТ 4

Содержание 5

Введение 8

1 Обзор предметной области 10

1.1 Постановка задачи 10

1.2 Инструменты для реализации клиента 11

1.2.1 Анализ требований 11

1.2.2 Язык программирования для реализации клиента 12

1.2.3 Win Forms API для создания графического интерфейса 13

1.2.4 Реализация пользовательского интерфейса 14

1.3 Инструменты для реализации сервера 15

1.3.1 Анализ требований 15

1.3.2 Windows служба как способ реализации сервера 17

1.3.3 Языка программирования для реализации сервера 20

1.4 Механизма для взаимодействия клиента и сервера 21

2 Реализация клиент-серверного приложения 23

2.1 Описание работы клиент-серверного приложения 23

2.1.1 Описание работы клиента 23

2.1.2 Описание работы сервера 24

2.2 Разработка серверной части приложения 26

2.3 Разработка клиентской части приложения 29

3 Описание приложения 34

3.1 Руководство пользователя 34

3.2 Спецификация сервера 42

3.2.1 Модуль Task 42

3.2.2 Модуль Action 44
3.2.3 Модуль Trigger 45
3.2.4 Модуль Registry 45
3.2.5 Модуль Service 46
3.2.5 Модуль RequestHandler 47
3.2.6 Модуль Message 48
3.2.7 Модуль Security 49
3.2.8 Модуль Time 49
3.2.9 Модуль GenericContainers 50
3.3 Спецификация клиента 53
3.3.1 Классы модели 53
3.3.2 Класс контроллера 57
3.3.3 Классы представления 58
3.3.4 Классы форм 60
3.3.5 Интерфейс с севером 60
Заключение 62
Список используемых источников 64
Приложение А Листинги программы клиента 65
Приложение А.1 Файл «MainForm.cs» 65
Приложение А.2 Файл «TaskController.cs» 69
Приложение А.3 Файл «TriggerView.cs» 74
Приложение А.4 Файл «TimeTrigger.cs» 79
Приложение А.5 Файл «Task.cs» 83
Приложение А.6 Файл «Action.cs» 100
Приложение А.7 Файл «Trigger.cs» 102
Приложение А.8 Файл «ServerWorker.cs» 114
Приложение А.9 Файл « Program.cs» 122
Приложение В Листинги программы сервера 123

Приложение B.1 Файл «Service.c» 123
Приложение B.2 Файл « Task.c» 127
Приложение B.3 Файл «Time.c» 149
Приложение B.4 Файл «Action.c» 153
Приложение B.5 Файл « RequestHandler.c» 155
Приложение C Результаты работы программы 175
Приложение C.1 – Вид начально экрана 176
Приложение C.2 – Вид экрана создания задачи 176
Приложение C.3 – Вид экрана с настройками действия 177
Приложение C.4 – Вид экрана настроек триггера 178
Приложение C.5 – Вид экрана подключения к серверу 178
Приложение C.6 – Вид экрана редактирования задачи 179
Приложение C.7 – Вид окна с информационным сообщением 179
Приложение D UML-диаграммы приложения 180
Приложение D.1 – UML-диаграмма классов клиента 180
Приложение D.2 – UML-диаграмма вариантов использования сервера 181
Приложение D.3 – UML-диаграмма вариантов использования клиента 181
Приложение D.4 – UML-диаграмма деятельности сервера 182
Приложение D.5 – UML-диаграмма деятельности клиента 183
Приложение D.6 – UML-диаграмма последовательности 184
Приложение D.7 – UML-диаграмма развертывания 185
Приложение E Презентация 186

Введение

Современные сайты, для поддержания бесперебойной работы, могут располагаться на нескольких серверах. Это позволяет предотвратить отказ в обслуживании при пиковых нагрузках. В то же время высоконагруженные сервисы могут задействовать сотни или тысячи вычислительных машин, располагающиеся в огромных дата-центрах.

Широкое распространение приобрело использование облачных технологий для высоконагруженных и распределённых систем, которые образуют совокупность компьютеров, предоставляющих доступ к своим вычислительным ресурсам подобно одиночной системе.

ЭВМ требуют, как аппаратной, так и программной поддержки. Непрерывную бесперебойную работу обеспечивают системные администраторы. В их обязанности входит слежение и обеспечение безотказной работы системного программного обеспечения, проведение резервного копирования данных, а также их восстановление при необходимости. Ручной подход решения этих задач требует либо огромного количества времени, либо содержание большого штата администраторов. Что увеличивает расходы на поддержание работоспособности системы.

Решением данной проблемы является использования программных продуктов для автоматизации выполнения различных действий, с учетом происходящих в системе событий. Такие программы называются планировщиками задач. Их применение позволяет добиться следующих преимуществ.

Снижение затрат. Поскольку бюджеты ИТ постоянно находятся под давлением, ключевым фактором является сокращение операционных расходов. Планировщики задач сокращают эксплуатационные расходы за счет сокращения количества персонала, необходимого для выполнения

определенных задач. В то же время, планирование работы позволяет предприятиям более эффективно использовать инфраструктуру. Например, большинство современных серверов имеют низкие эксплуатационные расходы, и общая стоимость владения снижается. Тем не менее, стоимость операций персонала может составлять до 71% от общего объема расходов [6]. С планировщиком задач сохраняется больше от этих активов, что значительно улучшает рентабельность инвестиций.

Производительность. Программное обеспечение планирования задач увеличивает пропускную способность за счет автоматизации расписания производства. Время и деньги сохраняются путем устранения времени задержки между заданиями и минимизации вмешательства оператора.

Надежность. Автоматическое планирование гарантирует, что задания не будут забыты. Оно следит за успешным завершением задач, что входные данные верны и выполняется вся указанная обработка.

Целью данной работы является разработка клиент-серверного приложения для планирования выполнения задач на удаленных и локальных вычислительных машинах. Клиентская часть отвечает за создание и редактирование задач, а также для отображения уже существующих. Сервер должен обеспечивать возможность работы одновременно с несколькими клиентами. Запросы от клиентов обрабатываются последовательно и независимо друг от друга. При наступлении заданных событий должны быть запущены действия, указанные в задаче.

1 Обзор предметной области

1.1 Постановка задачи

Требуется разработать клиент-серверное приложение, позволяющее планировать выполнение задач на удаленных и локальных компьютерах по расписанию.

Приложение должно отвечать следующим требованиям:

1. Планировщик задач должен предоставлять возможность пользователю создавать, изменять и удалять задачи.
2. Каждая задача должна иметь свой список действий, которые она исполняет.
3. Каждая задача должна иметь свой список триггеров, которые инициируют выполнение задачи.
4. Каждая задача должна иметь информацию о пользователе, в контексте которого будет осуществляться выполнение задачи.
5. Клиентская часть должна давать возможность отображать список имеющихся задач.
6. Клиентская часть должна давать возможность подключаться к удаленной машине для редактирования её задач.
7. Планировщик задач должен работать в ОС Microsoft Windows 7 и выше.

Действие – это запуск программы или исполнение скрипта, указываемое пользователем.

Триггер – это событие, наступление которого инициирует выполнение задачи. Необходимо создать набор различных типов триггеров, а также настройки по их автоматической деактивации. К ним относятся следующие типы триггеров:

1. Однократный триггер – срабатывает при наступлении указанной даты и времени.
2. Ежедневный триггер – срабатывает через указанный промежуток в днях при наступлении указанного времени.
3. Еженедельный триггер – срабатывает через указанный промежуток в неделях, в указанные дни недели, при наступлении установленного времени.
4. Ежемесячный триггер – срабатывает либо в указанные дни месяца, либо в указанные дни недель установленных месяцев.

Различные типы триггеров, а также их сочетания в задачах позволяют строить сложные и гибкие условия выполнения задач.

1.2 Инструменты для реализации клиента

Клиентское приложение – то с чем непосредственно взаимодействует пользователь. Инструменты его реализующие, должны позволять создать интерфейс, обеспечивающий лучший опыт взаимодействия с ним (user experience).

1.2.1 Анализ требований

Современное приложение должно иметь пользовательский интерфейс, выполненный в современном дизайнерском стиле с использованием спокойных цветов. В дополнение, он также должен иметь грамотное расположение элементов. Что позволит пользователю легко ориентироваться в интерфейсе. Следует принимать во внимание группы пользователей с различными ограничениями связанные с заболеваниями. Необходимо продумать и реализовать инструменты, с помощью которых они бы смогли полноценно использовать все возможности приложения наравне с другими.

Интегрированная среда разработки должна поддерживать средства отображения и редактирования графической части, написанной на выбранном языке программирования. Это обеспечит не только удобство при разработке и

дальнейшей поддержки приложения, но и позволит в дальнейшем вносить в него изменения человеку без навыков программирования.

Не маловажным является выбор языка программирования, на котором будет написано приложение. Необходимо чтобы он позволял создавать сущности высокого уровня, поддерживал модульность и встроенный событийный аппарат. Эти возможности позволяют создать гибкое приложение со слабо связанной графической частью и бизнес моделью. Это обеспечит высокий уровень расширяемости и масштабирования приложения.

1.2.2 Язык программирования для реализации клиента

Учитывая требования, которые описаны в предыдущем пункте, языком программирования для реализации клиента был выбран C#. Это высокоуровневый, мультипарадигменный язык программирования. В нем реализованы: строгая типизация – не позволяет обращаться к неинициализированным переменным, выходить за пределы массива или выполнять неконтролируемое приведение типов; сборка мусора – автоматически освобождает память, занятую уничтоженными и неиспользуемыми объектами; обработка исключений дает структурированный и расширяемый способ выявлять и обрабатывать ошибки; императивный, функциональный, а так декларативный стиль; объектно-ориентированный и компонентно-ориентированный подходы. В нем реализована нативная поддержка событий, что позволяет легко создавать слабосвязанные и реализовывать повторно водимые модули, имеющие огромное значение в параллельном программировании.

Помимо основных принципов ООП (инкапсуляция, наследование, полиморфизм), C# поддерживает ряд особенных языковых конструкций, облегчающих разработку компонентов.

- Делегаты – позволяет инкапсулировать сигнатуру методов, для реализации типобезопасных уведомлений о событиях.
- Свойства – выполняют функции аксессоров для приватных членов.

– Атрибуты – предоставляют декларированные метаданные о типах во время исполнения.

Разработка современных приложений все больше тяготеет к созданию программных компонентов в форме автономных и самоописательных пакетов, реализующих отдельные функциональные возможности. Важная особенность таких компонентов — это модель программирования на основе свойств, методов и событий. Каждый компонент имеет атрибуты, предоставляющие декларативные сведения о компоненте, а также встроенные элементы документации. C# предоставляет языковые конструкции, непосредственно поддерживающие такой принцип работы. Благодаря этому он отлично подходит для создания и применения программных компонентов.

1.2.3 Win Forms API для создания графического интерфейса

Приложения с графическим интерфейсом могут быть созданы на языке C# с использованием Windows Forms. Windows Forms – это способ конструирования приложений на базе классов .NET Framework class library. В них используется собственная модель программирования, которая совершеннее, чем основанные на Win32 API или MFC. Win Forms приложения выполняются в управляемой среде .NET Common Language Runtime (CLR).

Главная выгода от написания Windows-приложений с использованием – Windows Forms – то, что он гомогенизирует (строит однородную структуру) программную модель и устраняет многие ошибки и противоречия, возникающие при работе с Windows API.

NET Framework class library содержит намного больший набор классов для реализации графической части любой сложности. Приложения, написанные с использованием Windows Forms, содержат меньше кода, чем аналогичные, но использующие Windows API или MFC.

Другая выгода от Windows Forms – это наличие идентичного API для набора разных языка программирования. Все приложения, которые используют Windows Forms, используют один API из .NET Framework class library. Знание

одного API позволяет программисту писать приложения фактически на любом языке, поддерживаемый .NET Framework.

1.2.4 Реализация пользовательского интерфейса

Графические пользовательские интерфейсы стали привычной частью программных продуктов, как для пользователей, так и для разработчиков. При их проектировании, возникают определенные задачи и проблемы в разработке системы. Что привело к созданию ряда различных, но подобных решений. Одним из которых является шаблон MVC.

Model View Controller (MVC) – шаблон проектирования основной идеей которого является разделение данных приложения, пользовательского интерфейса и логики его работы на отдельные компоненты: модель, представление и контроллер – таким образом, чтобы изменение отдельного компонента не затрагивало другие.

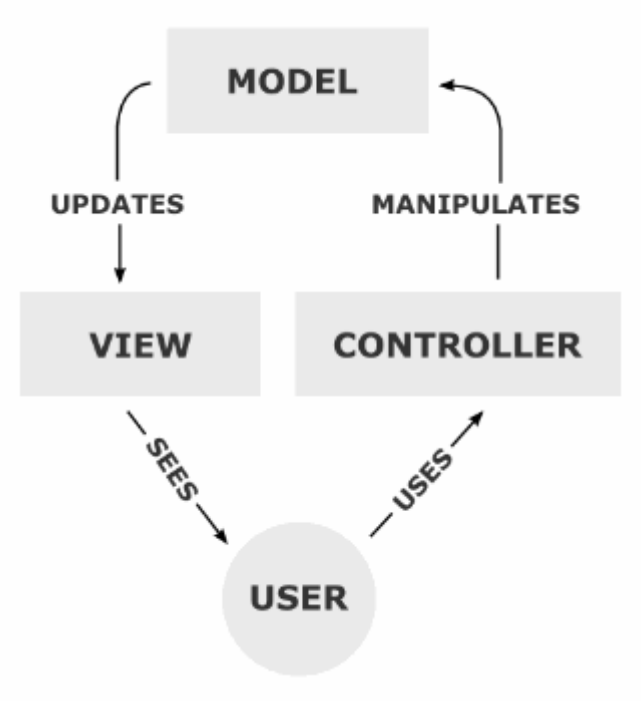


Рисунок 1 – Шаблон проектирования MVC.

Модель. Объекты модели – являются частью приложения, отвечающие за логику программы. Часто состояния объектов модели сохраняются в базе данных.

Представление – это компоненты, отображающие пользовательский интерфейс приложения. Как правило, именно пользовательский интерфейс отображает данные модели.

Контроллер – это компоненты, которые обрабатывают взаимодействие пользователя, работают с моделью и в конечном итоге выбирают представление для визуализации, отображающее пользовательский интерфейс. В MVC приложении, представление отображает информацию; контроллер обрабатывает и отвечает на ввод и взаимодействие пользователя.

Шаблон MVC помогает создавать приложения, которые разделяют различные аспекты приложения (логика ввода, бизнес-логика и функционирование пользовательского интерфейса), обеспечивая при этом слабую связь между ними. MVC устанавливает между компонентами протокол взаимодействия “подписка/оповещение”. Представление гарантирует, что оно отображает состояние модели. При каждом изменение внутренних данных модель оповещает все зависящие от неё представления, в результате чего они обновляют себя [3, С. 18]. Функционал пользовательского интерфейса принадлежит к представлению. Входная логика принадлежит контроллеру. Бизнес-логика принадлежит модели. Это разделение помогает управлять сложностью при создании приложения, поскольку оно позволяет сосредоточиться на одном аспекте реализации за раз.

1.3 Инструменты для реализации сервера

Сервер – это компьютерная программа или устройство выполняющие услуги для других программ или устройств, которые называются клиентами. При выборе технологий для реализации сервера нужно учитывать их влияние на быстродействие, отказоустойчивость и масштабируемость.

1.3.1 Анализ требований

Сервер должен выполнять различные функции: хранение данных, их распределение между клиентами, выполнение с ними различных вычислений.

Один сервер может обслуживать несколько клиентов, а один клиент может использовать несколько серверов. Часто сервер воспринимается, как удаленный компьютер, обслуживающий запросы. На самом деле клиентский и серверный процессы могут выполняться на одном устройстве.

С понятием сервер тесно связанна клиент-серверная модель, на данный момент являющаяся наиболее распространённой и используемая во многих программных решениях. Она характеризуется гибкостью и простотой использования. Наиболее часто в клиент-серверной модели используется протокол согласно которому, клиент посылает запрос на сервер. Он разбирает запрос, выполняет действия, указанные в нем и возвращает результат исполнения клиенту в виде ответа.

Существует множество типов серверов, несколько из них приведены ниже:

1. Сервер приложений – это программа на компьютере в распределенной сети, которая обеспечивает бизнес-логику для прикладной программы.
2. Прокси-сервер – это программное обеспечение, которое выступает в качестве посредника между конечным устройством, таким как компьютер, и другим сервером, с которого пользователь или клиент запрашивает услугу.
3. Почтовый сервер – это приложение, которое получает входящую электронную почту от локальных пользователей (людей в пределах одного домена) и отправителей, а затем посылает исходящую электронную почту.
4. Виртуальный сервер – это программа, работающая на общем сервере, которая настроена таким образом, что каждому пользователю кажется, что у них есть полный контроль над сервером.
5. Блейд-сервер – это серверное шасси с множеством тонких, модульных электронных плат, известных как blade-серверы. Каждый blade-сервер

является отдельным сервером, часто предназначенным для одного приложения.

6. Файловый сервер – это компьютер, отвечающий за централизованное хранение и управление данными, чтобы другие компьютеры в одной сети могли получить к ним доступ.
7. Сервер политики – это компонент безопасности сети на основе политик, которая предоставляет услуги авторизации и облегчает отслеживание и управление доступом к данным.

Сервер должен осуществлять следующие действия: обработка входящих запросов от клиентов, выдача ответа о проделанной работе, проверка прав клиентов. Он может располагаться на том же компьютере, что и клиент либо удаленно.

1.3.2 Windows служба как способ реализации сервера

Почти в каждой операционной системе есть механизм, запускающий при старте самой системы те процессы, которые предоставляют службы, не связанные с интерактивным пользователем. В ОС Microsoft Windows такие процессы называются службами или Windows-службами потому, что они для взаимодействия с системой используют API-функции Windows. Службы похожи на имеющиеся в UNIX процессы-демоны и часто реализуются на серверной стороне клиент-серверных приложений. В качестве примера Windows-службы можно назвать веб-сервер потому, что он должен быть запущен независимо от того, зарегистрировался ли кто-нибудь на компьютере или нет, и должен быть запущен при старте системы, чтобы администратору не нужно было даже помнить о необходимости его запуска или запускать его самостоятельно. Windows-службы состоят из трех компонентов: приложения службы, программы управления службой — service control program (SCP) — и диспетчера управления службами — service control manager (SCM) [2, С. 364].

1.3.2.2 Жизненный цикл службы

Служба проходит через несколько состояний за всю свою “жизнь”. В начале, служба должна быть установлена в систему, на которой она будет работать. Для этого она должна быть установлена с помощью диспетчера управления службами. Диспетчер управления службами является центральной утилитой, предоставляемой Windows для администрирования служб.

Служба может существовать в одном из трех основных состояний: “Работает”, “Приостановлена” или “Остановлена”. Она также может сообщать о состоянии ожидающей команды: “Ожидание продолжения”, “Ожидание паузы”, “Ожидание старта” или “Ожидание остановки”. Эти статусы указывают на то, что команда была вызвана, например, команда для приостановки работы службы, но еще не выполнена. Можно запросить статус, чтобы определить, в каком состоянии находится служба, или использовать WaitForStatus для ожидания наступления конкретного состояния.

После того, как служба была установлена, ее необходимо запустить. Запуск позволяет ей начать работу. Он может быть осуществлен из диспетчера управления службами, из проводника сервера или из кода.

Можно приостановить, остановить или возобновить работу службы из диспетчера служб, из проводника сервера или путем вызова функции в коде. Каждое из этих действий может вызвать связанную процедуру в службе (OnStop, OnPause или OnContinue), для осуществления дополнительной обработки, которая будет выполняться при изменении состояния службы.

1.3.2.3 Отличия между службой и обычным приложением

Механизм запуска – конечный пользователь запускает обычное приложение с рабочего стола или меню “Пуск”. Службы запускаются сразу после запуска ОС до момента входа пользователя в систему.

Несколько экземпляров – только один экземпляр службы может быть запушен на устройстве. Обычные приложения могут иметь несколько.

Пользовательский интерфейс – в отличие от обычных приложений службы не имеют пользовательского интерфейса, они запускаются в фоновом режиме, и пользователь не взаимодействует с ними напрямую. Службы не останавливаются, когда пользователь выходит из системы, в то время как обычное приложение завершит свою работу.

Права администратора – службы запускаются с правами администратора.

1.3.2.4 Типы служб

Существует два типа служб, которые могут быть. Службе, являющейся единственной в процессе, присваивается тип Win32OwnProcess. Если несколько служб, совместно используют один и тот же процесс, им присваивается тип Win32ShareProcess.

1.3.2.5 Примеры служб в Windows

Active Directory Service – это служба разработанная Microsoft для сетей Windows. Она констатирует централизованное управление доменами и связанными с ними функциями

Prefetch and Superfetch Service – это служба ускоряет работу операционной системы и приложений путем кэширования в RAM часто используемых файлов, библиотек и компонентов приложений.

Internet Connection Sharing Service – служба позволяет использовать устройство подключенное к Интернету, в качестве точки доступа для других устройств. Доступ может осуществлять через широкополосную сеть Ethernet, сотовую службу или другие шлюзы.

Routing and Remote Access Service – эта служба позволяет создавать приложения с возможностями удалённого доступа к маршрутизации. Что позволяет машине работать в качестве сетевого маршрутизатора.

Разработчики часто используют службы для реализации программ требующих работы в фоновом режиме, например, для мониторинга различных

данных о системе и своевременного реагирования на заданные пороговые значения.

На основе всей вышеизложенной информации, служба полностью подходит для реализации на её основе сервера.

1.3.3 Языка программирования для реализации сервера

Сервер является фундаментом любого клиент-серверного приложения. Множество различных факторов влияют на его конечное качество, начиная от выбора архитектуры заканчивая аппаратной составляющей. Однако выбор языка программирования является определяющим. От него зависит быстродействие сервера, его масштабируемость, а также скорость разработки и удобство программирования.

Для его реализации был выбран язык программирования Си. Си – универсальный, императивный язык программирования, поддерживающий структурное программирования. Дизайн языка, его конструкции близки типичным машинным инструкциям, благодаря чему Си находит широкое применение в проектах, для которых свойственен язык ассемблера, например, написание операционных систем. Он используется при написании прикладного программного обеспечения, предназначенного для широкого набора типов устройств начиная от встраиваемых систем до суперкомпьютеров. Си обладает следующими особенностями:

- простая языковая база, из которой вынесены в библиотеки многие существенные возможности, вроде математических функций или функций управления файлами;
- ориентация на процедурное программирование, обеспечивающая удобство применения структурного стиля программирования; система типов, предохраняющая от бессмысленных операций;
- использование препроцессора для определения макросов и включения файлов с исходным кодом;

– непосредственный доступ к памяти компьютера через использование указателей;

– минимальное число ключевых слов;

– передача параметров в функцию по значению, а не по ссылке (при этом передача по ссылке эмулируется с помощью указателей);

– указатели на функции и статические переменные;

– области действия имён;

– структуры и объединения – определяемые пользователем собирательные типы данных, которыми можно манипулировать как одним целым;

1.4 Механизма для взаимодействия клиента и сервера

Клиент-серверное взаимодействие может осуществляться с использованием различных механизмов. Одни напрямую используют транспортный сетевой протокол, другие строят различные абстракции позволяющие, например, буферизировать сообщения. Исходя из выбранной операционной системы и требований безопасности, в качестве механизма взаимодействия выбран именованный канал.

Именованный канал является односторонним или дуплексным каналом, используемый для связи между серверным, одним или несколькими клиентскими каналами. Все экземпляры именованного канала имеют одно и то же имя, но каждый экземпляр имеет свои собственные буферы и дескрипторы, и предоставляет отдельный канал для клиент-серверного взаимодействия.

Использование экземпляров позволяет нескольким клиентам одновременно использовать один и тот же канал. Любой процесс может обращаться к именованным каналам, при условии проверки безопасности,

делая именованные каналы простой формой связи между связанными или несвязанными процессами. Любой процесс может действовать и как сервер и как клиент, что делает возможным одноранговую связь. Используемый здесь термин “сервер трубопроводов” относится к процессу, который создает именованный канал. Термин “клиент” относится к процессу, который соединяется с экземпляром именованного канала.

Именованные каналы могут использоваться для обеспечения связи между процессами на одном компьютере или между процессами на разных компьютерах по сети. Если служба сервера запущена, все именованные каналы доступны удаленно. Для отключения межсетевого взаимодействия нужно запретить доступ к NT AUTHORITY \ NETWORK или переключиться на локальный RPC.

В соответствии с информацией, представленной выше, именованные каналы подходят для организации межсетевого взаимодействия клиента и сервера.

2 Реализация клиент-серверного приложения

В данной главе описана реализация клиентской и серверной частей приложения. Особое внимание уделяется выделению и описанию различных сущностей, и их взаимодействию между собой. При этом освещены основные особенности реализации, связанные с программированием на выбранных языках. Целью данного раздела является объяснить принцип работы и внутреннее устройство системы в целом.

2.1 Описание работы клиент-серверного приложения

Клиент серверное приложение включает в себя клиент и сервер. Клиент – это программа с графическим интерфейсом предназначенная для осуществления взаимодействия с пользователем для исполнения его команд. Сервер – это программа, работающая в фоновом режиме на целевом компьютере и непосредственно осуществляющая планирование и выполнение задач. Клиент и сервер осуществляют общение по именованному каналу.

2.1.1 Описание работы клиента

Клиент начинает свою работу, посылая запрос на сервер для получения списка задач. По умолчанию подключение осуществляется к локальному серверу. Далее пользователь может взаимодействовать с задачами, например, удалять, изменять их, а также создавать новые. Если локального сервера нет, то пользователю будет продемонстрированно сообщение о неудачной попытке подключения.

Создавая новую задачу, пользователю необходимо ввести все необходимые данные, послу чего клиент посылает соответствующий запрос на сервер. В случаи возникновения ошибки при обработке запроса, клиенту возвращается информация о ней, обработав которую клиент выводит

информационное сообщение. При удалении и изменении задачи выполняется та же последовательность действий.

Подключаясь к другому серверу, пользователь указывается его ip-адрес или имя компьютера в сети, после чего производится попытка подключения к нему. В случае успеха с сервера приходит ответ, содержащий список его задач. При провале подключения выводится соответствующее информационное сообщение.

2.1.2 Описание работы сервера

Сервер начинает работу с начальной инициализации параметров и подсистем, необходимых для его нормальной работы.

Обработкой запросов занимается отдельная нить сервера. После подключения клиента, сервер считывает информацию о запросе и его данные. Далее производится блокировка нити обработки запуска задач, обеспечивая тем самым взаимоисключение нитей при обращении к разделяемым ресурсам. Затем сервер выполняет действия, запрашиваемые клиентом, после чего снимает блокировку. Сервер может выполнять следующие действия.

Добавление задачи. Осуществляется попытка её записи в хэш таблицу, установка её триггеров, а также запись в реестр. В случаи возникновения ошибки сервер уведомляет о ней клиента.

Удаление задачи. Производится удаления записи из хэш таблицы, сброс триггеров удаляемой задачи, а также удаление записи из реестра. В случае возникновения ошибки сервер уведомляет о ней клиента.

Изменение задачи. Реализуется путем последовательного удаления старой версии задачи и добавления новой.

Получение списка задач. Сервер поочередно проходит по всем задачам и отсылает их клиенту.

Выполнением задач, при наступлении требуемых условий занимается отдельная нить сервера. Она ожидает наступления событий, указанных в

триггерах и запускает действия задачи при выполнении. Нить планирования осуществляет блокировку нити обработки запросов, для избегания коллизий в случаи одновременного изменения разделяемых нитями ресурсов.

На основе описания работы клиента и сервера можно составить UML-диаграмму последовательности (рисунок 2). Она визуализирует действия, описанные выше.

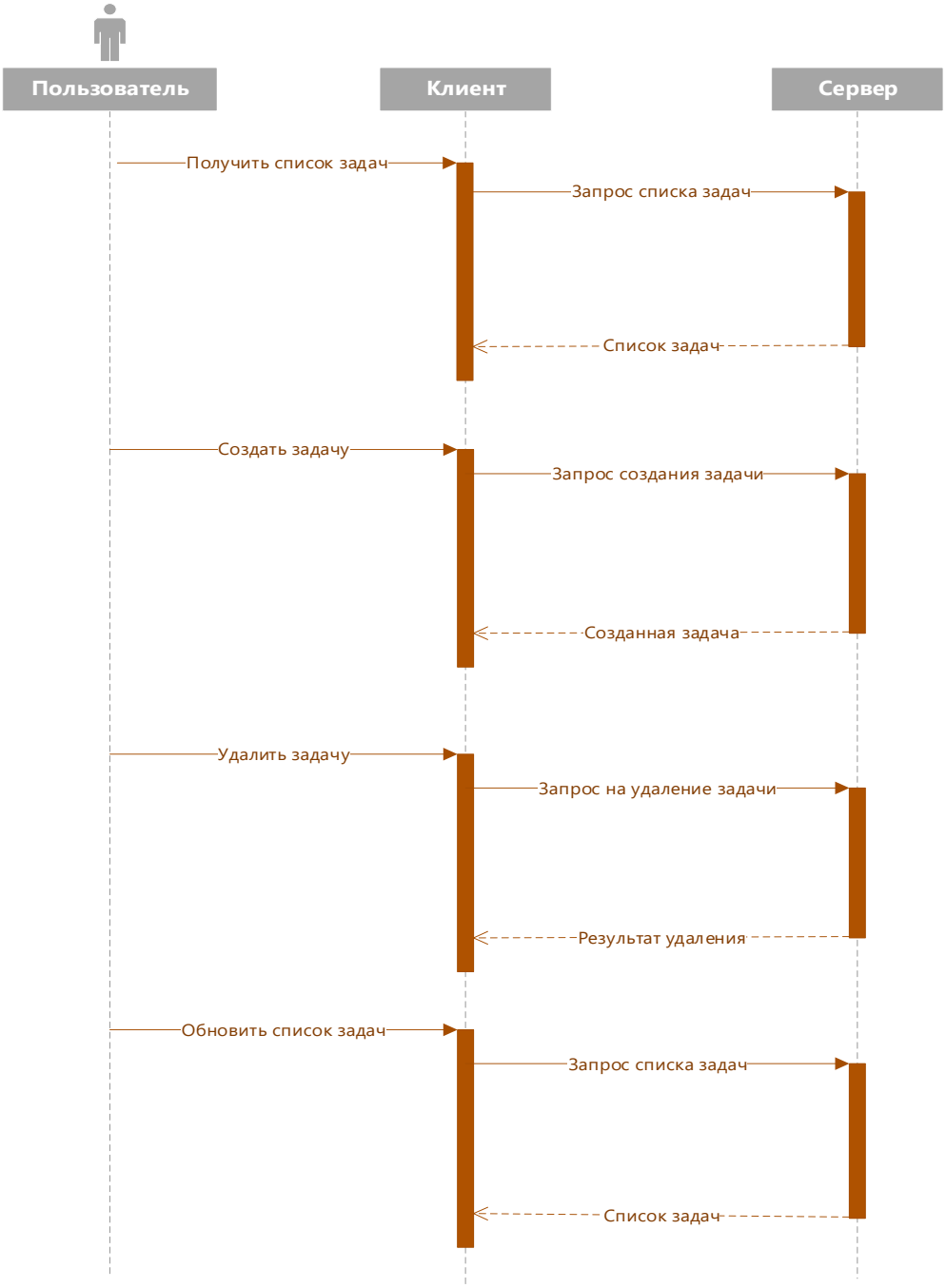


Рисунок 2 - UML-диаграмма последовательности

2.2 Разработка серверной части приложения

Сервер является той частью, которая отвечает за планирование выполнения задач. Исходя из специфики функционирования, он не имеет ни графического, ни консольного пользовательского интерфейса. Сервер работает в фоновом режиме. UML-диаграмма вариантов использования сервера представлена на рисунке 3.

Рисунок 3 – UML-диаграмма вариантов использования сервера

Разрабатываемый сервер состоит из двух основных подсистем. Первая отвечает за прием и обработку запросов клиента. Вторая планирует выполнение задач и запускает их при наступлении условий срабатывания. Каждая из них выполняется в отдельной нити. Следует более детально описать работу каждой из них.

Обработчик запросов ожидает подключение клиента по именованному каналу. Как только подключается новый клиент, сервер считывает информацию о запросе и данных переданных с ним. Далее происходит разбор его с целью определения запрашиваемого действия. Перед его выполнением обработчик

запрашивает блокировку, чтобы не допустить ошибок при обращении к разделяемым данным. Как только блокировка получена, сервер осуществляет исполнение запроса, он может осуществлять следующие действия: добавление, удаление, изменение задачи, а также считывание всех задач. Если в процессе возникнет ошибка, все изменения будут отменены, и её код будет передан клиенту. По завершении обработки, обработчик запросов снимает блокировку и ожидает новые запросы.

Добавление новой задачи сопровождается записью в реестр, регистрацией её триггеров и добавлением в хэш-таблицу. В реестре сохраняется информация о самой задаче, а также о всех её действиях и триггерах. При добавлении в хэш-таблицу производится проверка на уникальность, если добавляемая задача уже существует, произойдет ошибка.

Запрос на удаление осуществляет изъятие задачи из хэш-таблицы, удаление её триггеров и записей из реестра. В случаи отсутствия задачи в хэш-таблице в момент изъятия произойдёт ошибка.

Получение всех задач, сопровождается перечислением всех задач в хэш-таблице и отсылкой информации о каждой найденной клиенту.

Нить, отвечающая за планирование и запуск задач, ожидает наступление условий, указанных в их триггерах. При срабатывании триггера она ищет задачу, которой он принадлежит, после чего выполняет все действия, указанные в ней. Перед обращением к данным задачи, нить запрашивает блокировку, чтобы не допустить ошибок при обращении к разделяемым данным. Время срабатывания задачи фиксируется в реестре. После происходит повторное планирование исполненной задачи, в соответствии с условиями, указанными во всех её триггерах. В конце нить освобождает блокировку и начинает ожидать наступление нового события.

На основе вышеизложенного можно построить UML-диаграмму деятельности сервера, представленную на рисунке 4.

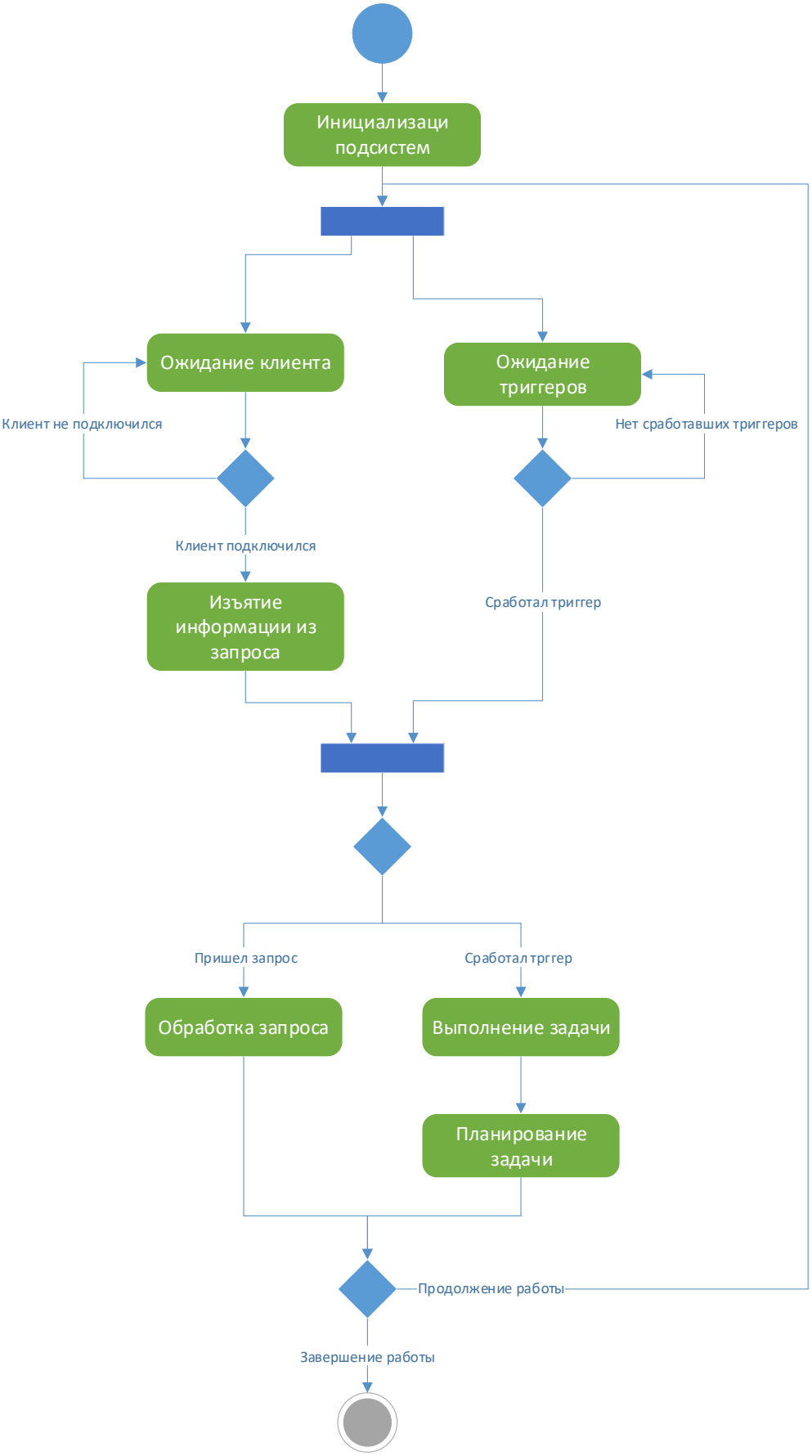


Рисунок 4 – UML-диаграмма деятельности сервера

Из диаграммы деятельности и описания работы можно выделить следующие состояния клиента:

1. Состояние ожидания клиента. В этом состоянии сервер ожидает подключения нового клиента.
2. Состояние обработка запроса. В этом состоянии сервер выполняет действия, запрашиваемые клиентом.
3. Состояние выполнения задачи. В этом состоянии сервер выполняет задачу.
4. Состояние отправки результата. В этом состоянии сервер отправляет результат обработки клиенту.
5. Состояние завершения работы. В этом состоянии сервер завершает работу.

Сервера реализуется на языке программирования Си, основным средством описания сущностей которого являются структуры. Работа со многими из них происходит по средствам вызова соответствующих функций, что позволяет структурировать код и уменьшить его повторяемость. Разделение кода, отвечающего за отдельные задачи, на отдельные файлы увеличивает читаемость и структурированность программы.

Немаловажной особенностью языка Си, которая помогает при работе с различными протоколами, это управление расположения структур в памяти программы. Это позволяет довольно просто организовать взаимодействие через различные механизмы, без накладных расходов на преобразования данных во время работы программы.

2.3 Разработка клиентской части приложения

Клиент является той частью приложения, которая осуществляет взаимодействие с пользователем. Поэтому он должен иметь графический пользовательский интерфейс, в котором пользователь может добавить новую задачу, удалить или редактировать уже существующую. На основе

вышеизложенного можно построить UML-диаграмму деятельности клиента, представленную на рисунке 5.

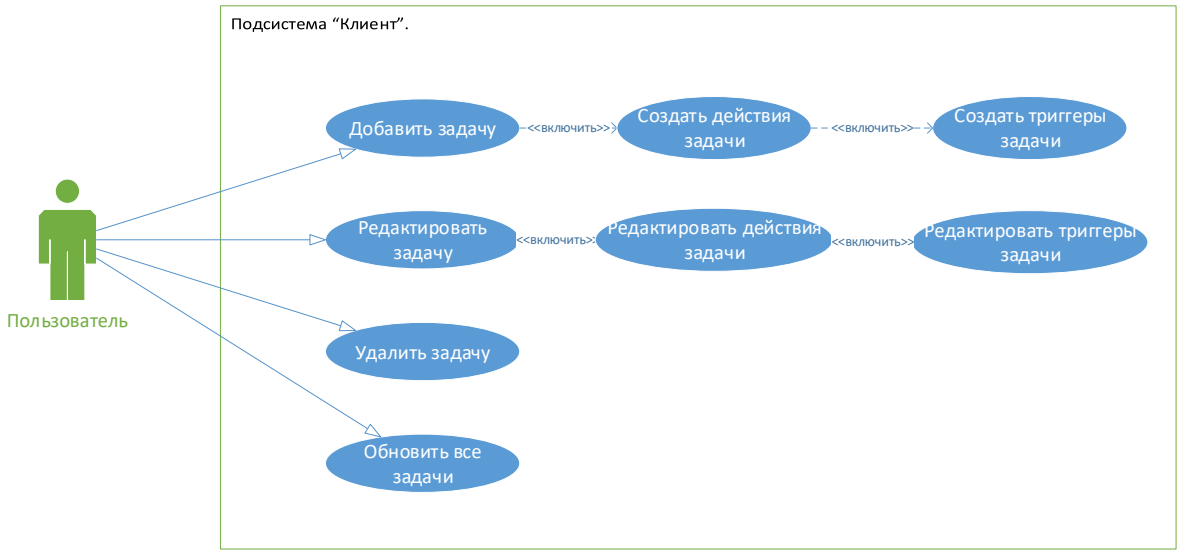


Рисунок 5 – UML-диаграмма вариантов использования клиента

Клиент начинает работу с попытки подключения к серверу и получения списка задач от него. После чего ожидает команду от пользователя. При добавлении новой задачи пользователь должен указать все необходимые параметры такие как: имя задачи, её описание, список действий и триггеров. Затем клиент отправляет запрос на сервер. В случае успеха задача отображается в общем списке. Если в процессе добавления произошла ошибка с сервера придёт информация о ней. И пользователю будет продемонстрированно соответствующее сообщение. Перед удалением существующей задачи, клиент посылает запрос на сервер, и в случае его успешного выполнения удаляет задачу на клиенте. При обновлении задач клиент запрашивает их новые версии с сервера, после чего заменяет ими старые. Подключение к удаленным серверам происходит по именованному каналу с указанием его ip-адреса на клиенте. В случаи успешного подключения клиент автоматически получает список задач с нового сервера.

На основе вышеизложенного можно построить UML-диаграмму деятельности клиента, представленную на рисунке 6.

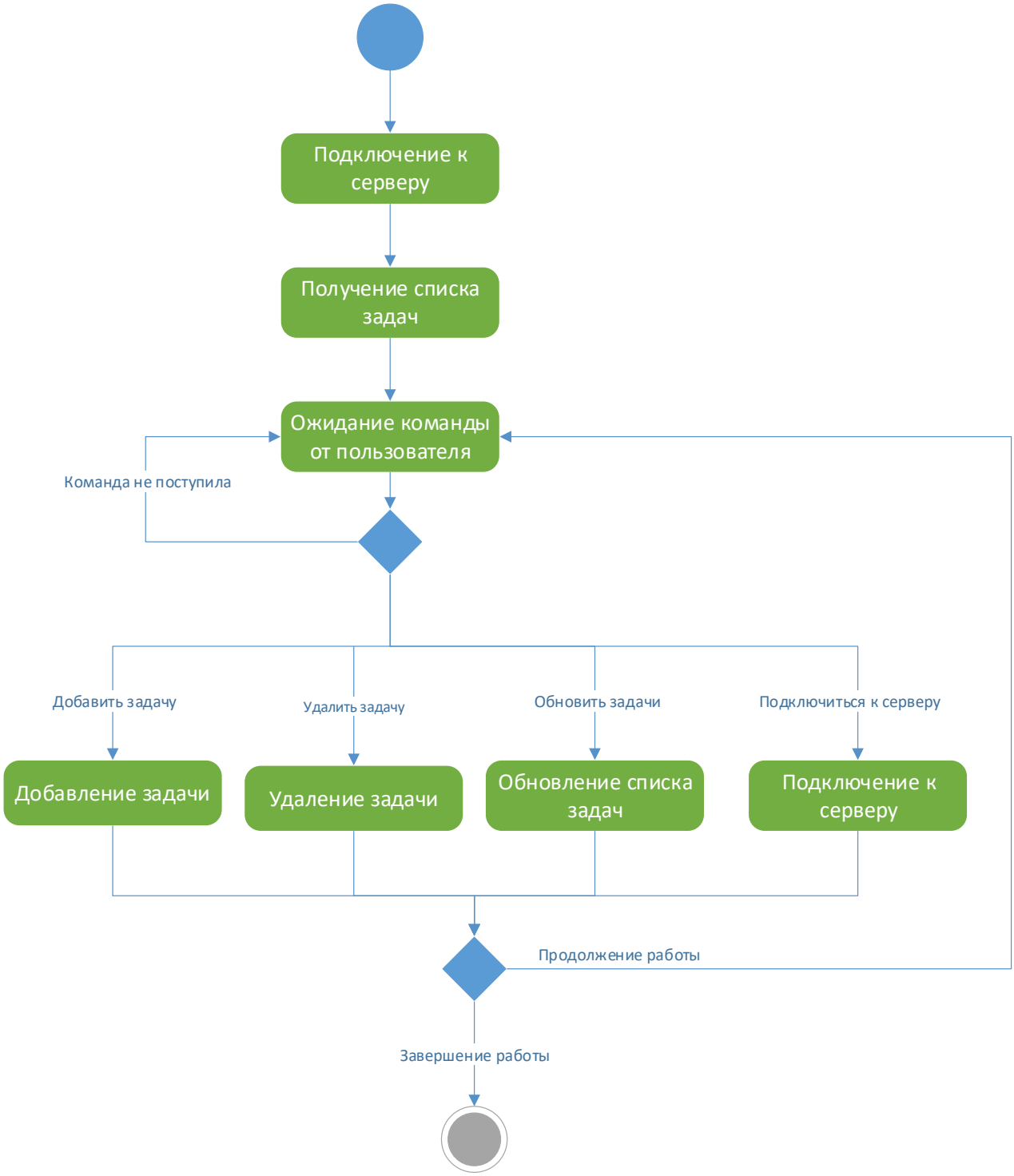


Рисунок 6 – UML-диаграмма деятельности клиента

Из диаграммы деятельности и описания работы можно выделить следующие состояния клиента:

1. Состояние подключение к серверу. В этом состоянии клиент пытается подключиться к серверу и получить список задач.

2. Ожидание команды от пользователя. В этом состоянии клиент ожидает получения команды от пользователя.
3. Состояние добавления задачи. В этом состоянии клиент обрабатывает запрос на добавление новой задачи.
4. Состояние удаление задачи. В этом состоянии клиент обрабатывает запрос на удаление существующей задачи.
5. Состояние обновление задач. В этом состоянии клиент осуществляет замену старых версий задач на новые, запрошенные с сервера.

Клиент разрабатывается на языке программирования C#. C# является объектно-ориентированным языком программирования, для описания сущностей в нем используются классы. Их использование позволяет создавать комплексные системы со сложной логикой. Рассмотрим классы, которые понадобятся для реализации клиента.

MainForm – это основной класс клиента. Данный класс отвечает за внешний вид главного экрана клиента. На нем расположены основные элементы взаимодействия пользователя с клиентом.

TaskController – это класс-контроллер. Данный класс отвечает за обеспечение взаимодействие классов моделей и классов представлений.

Task – это класс, который принадлежит модели в нем содержится вся информация о задаче.

Trigger – это класс, который содержит основную информацию о триггере задачи, является базовым для всех типов триггеров реализуемых в приложении.

Action – это класс, который содержит основную информацию о действии задачи, является базовым для всех типов действий реализуемых в приложении.

На рисунке 7 изображена UML-диаграмма классов клиента. Она показывает связи между классами.

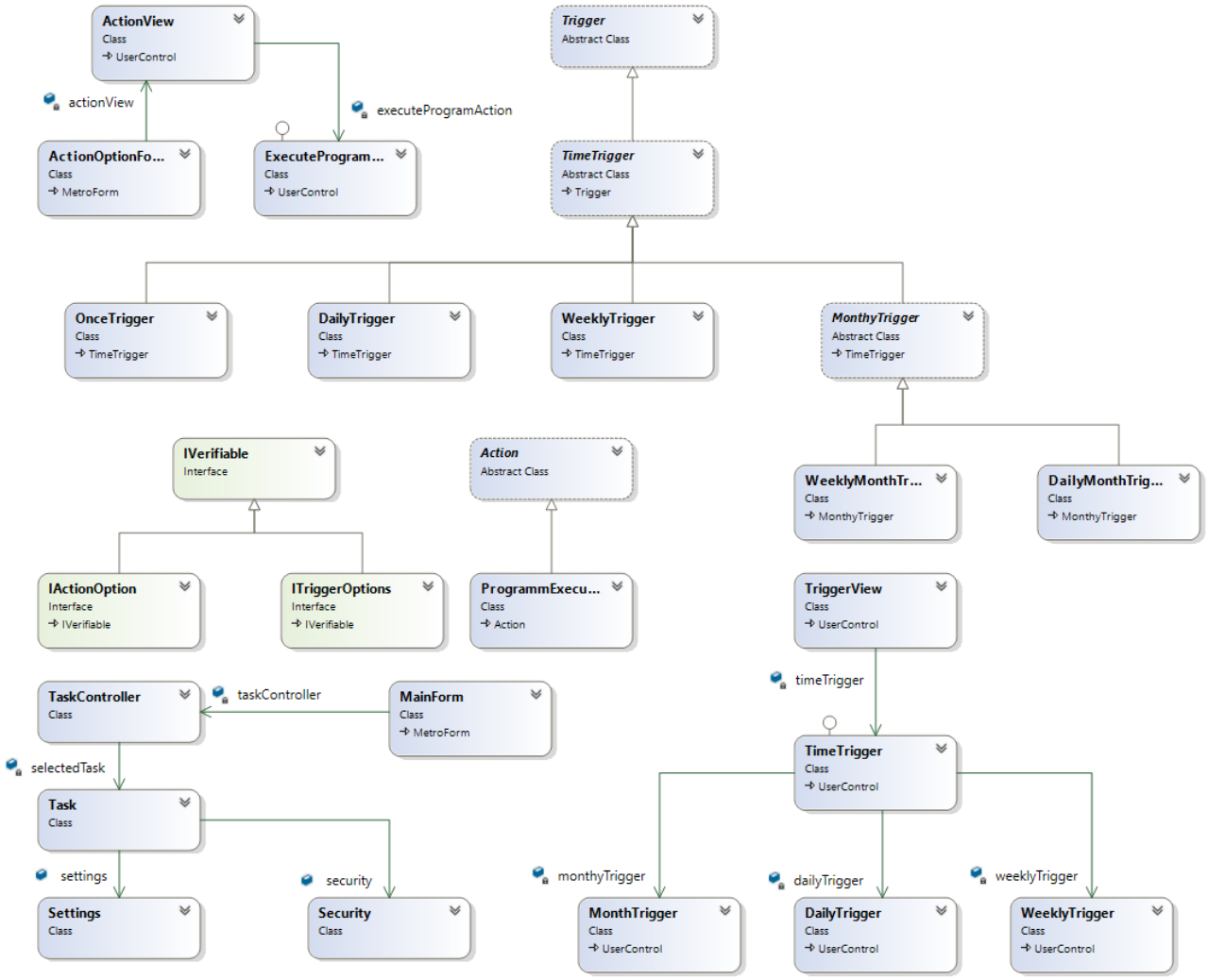


Рисунок 7 – UML-диаграмма классов клиента

3 Описание приложения

3.1 Руководство пользователя

После запуска клиента перед пользователем предстает главное окно программы (рисунок 8). На нем расположена таблица с задачами, а также кнопки для осуществления различных действий, таких как создание, редактирование, удаление задачи, обновление списка всех задач и подключение к серверу.

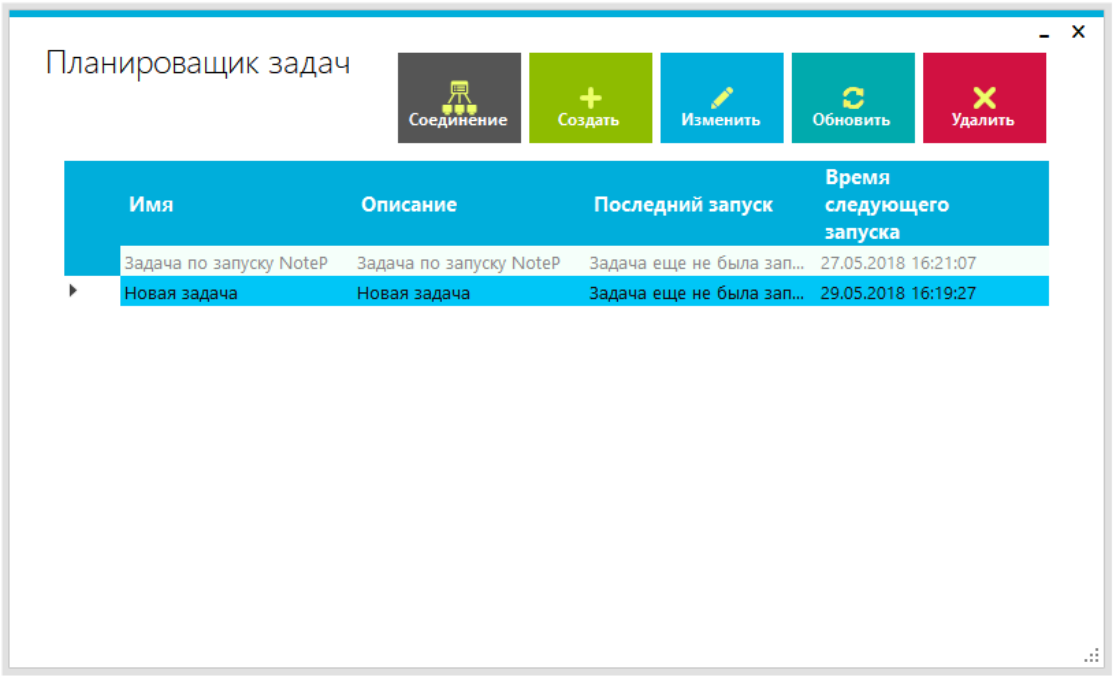


Рисунок 8 – Главная окно клиента

Чтобы создать новую задачу пользователю необходимо нажать на кнопку “Создать” (рисунок 8). После чего появится форма создания новой задачи (рисунок 9).

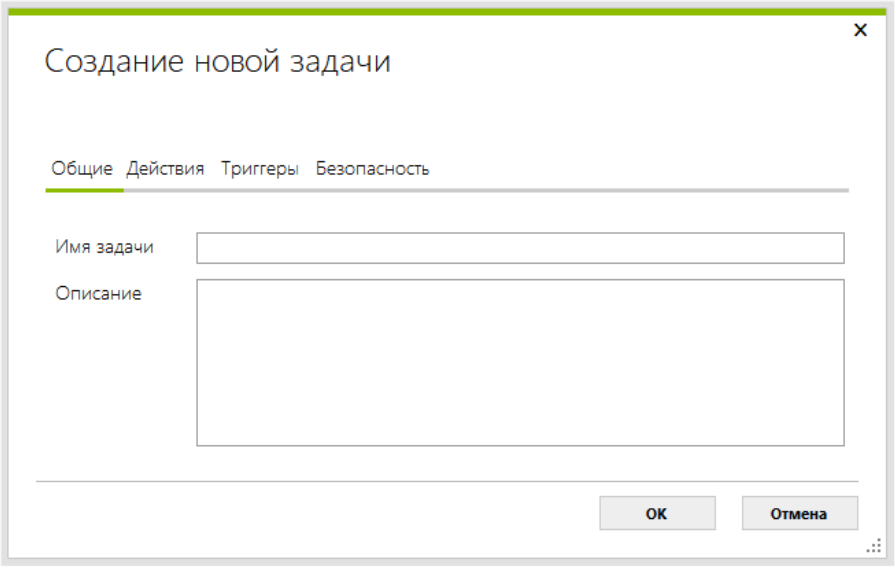


Рисунок 9 – Форма создания новой задачи. Вкладка “Общие”

На данной форме расположены несколько вкладок, каждая из которых группирует параметры задачи по определенным признакам. Во вкладке “Общие” задается имя и описание задачи (рисунок 9). Вкладка “Действия” содержит таблицу со списком действий выполняемых задаче (рисунок 10).

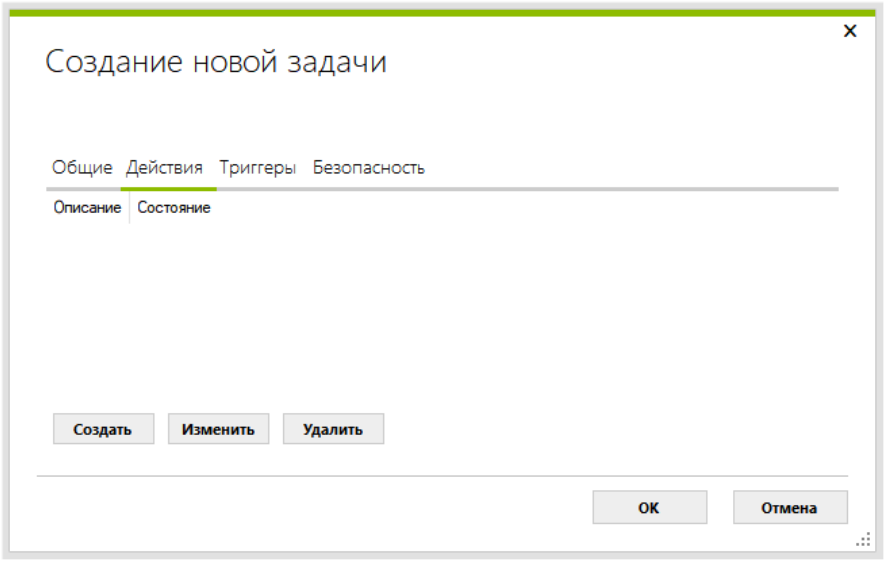


Рисунок 10 – Форма создания новой задачи. Вкладка “Действия”

Для создания действия задачи необходимо нажать на кнопку “Создать” (рисунок 10). После чего появится форма создания действия (рисунок 11). В ней нужно указать путь до программы или сценария, а также статус работы действия. Чтобы создать действие с указанными параметрами необходимо

нажать кнопку “Ок”, для отмены создания нежно нажать на кнопку “Отмена” (рисунок 11).

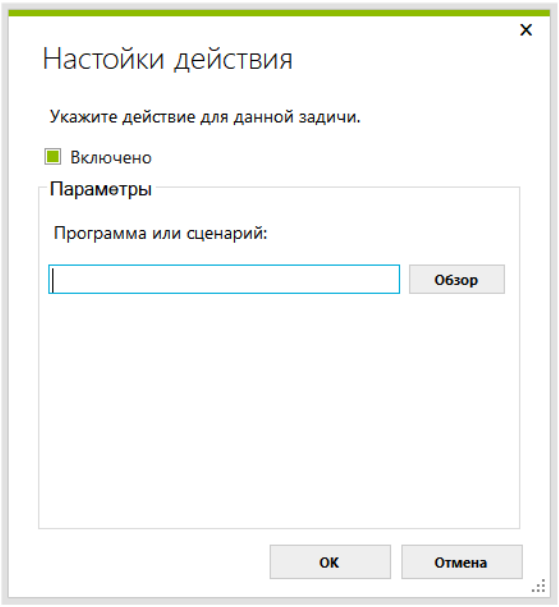


Рисунок 11 – Форма создания действия

Чтобы удалить созданное действие необходимо выбрать его в таблице действий и нажать на кнопку удалить “Удалить” (рисунок 10). Для изменения уже созданного действия необходимо выбрать его в таблице действий и затем нажать на кнопку “Изменить” (рисунок 10).

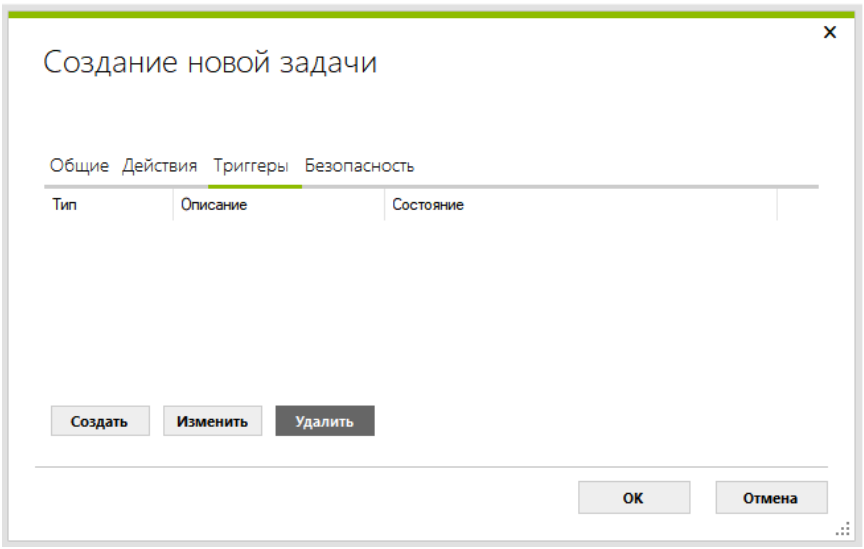


Рисунок 12 – Форма создания новой задачи. Вкладка “Триггеры”

Вкладка “Триггеры” содержит таблицу со списком триггеров задачи (рисунок 12). Для создания триггера задачи необходимо нажать на кнопку

“Создать” (рисунок 12). После чего появится форма создания триггера (рисунок 13).

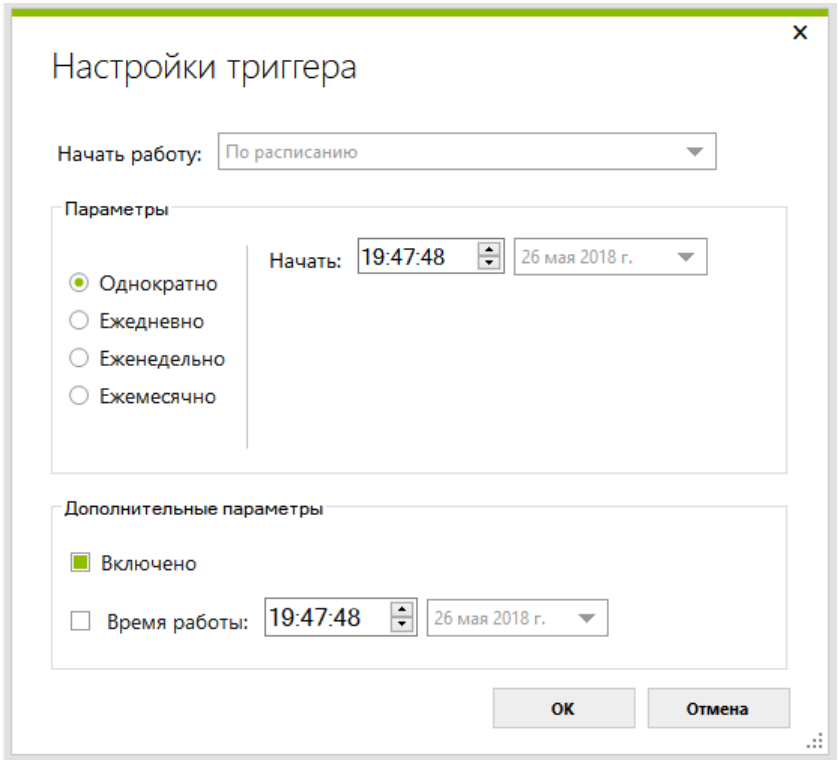


Рисунок 13 – Форма создания триггера

Триггер содержит множество различных настроек. Общие для всех триггеров располагаются в группе “Дополнительные параметры”, в ней находятся настройки времени и статуса работы триггера (рисунок 13). В группе “Параметры” расположены настройки специфичные для различных типов триггеров (рисунок 13).

Чтобы выбрать тип временного триггера необходимо нажать на один из переключателей: “Однократно”, “Ежедневно”, “Еженедельно”, “Ежемесячно” (рисунок 13). После выбора соответствующего типа, его параметры отображаются справа от вертикальной черты в группе “Параметры” (рисунок 13).

Однократный ежедневный триггер требует указания даты и времени его срабатывания (рисунок 13).

Еженедельный триггер требует указания даты и время его срабатывания, а также частоты повторения в днях (рисунок 14).

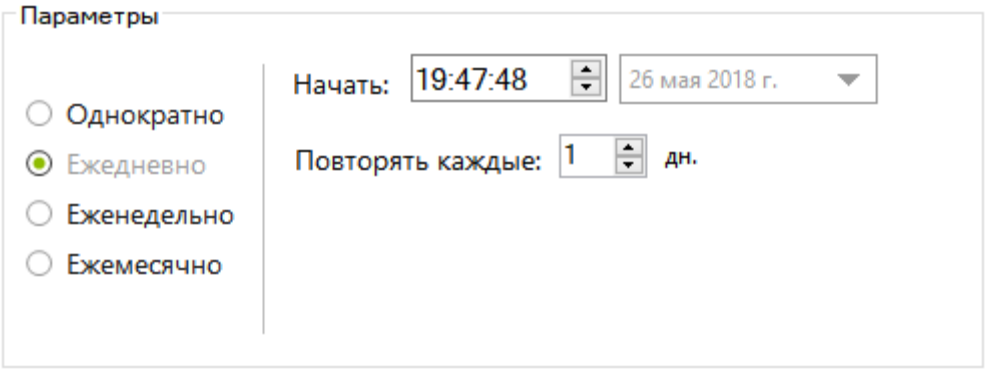


Рисунок 14 – Настройки еженедельного триггера

Еженедельный триггер требует указания даты и время его срабатывания, частоты повторения в неделях, а также дней недели (рисунок 15).

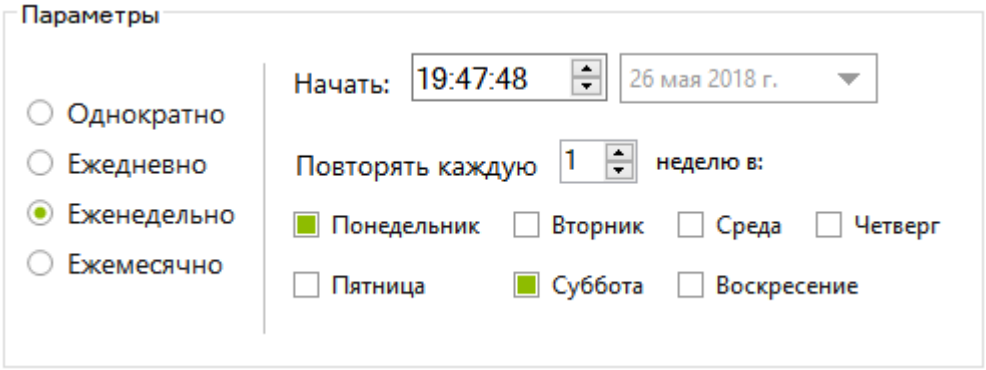


Рисунок 15 – Настройки еженедельного триггера

Ежемесячный триггер требует указания времени и даты срабатывания, месяца в году, а также параметров, зависимых от подтипа ежемесячного триггера, выбираемого переключателями: “Дни”, “В”. Первый определят месячно-дневной триггер, другой месячно-недельный триггер. Для первого подтип требует указания дней в месяце, второму нужно указать порядок и дней недели.

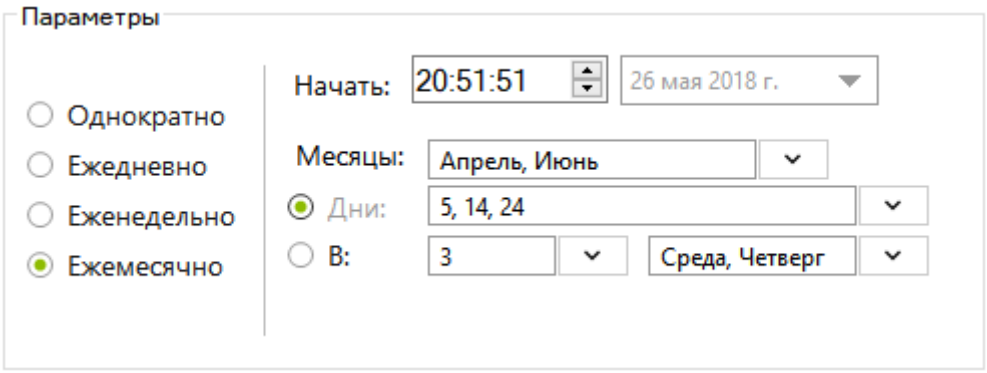


Рисунок 16 – Настройки ежемесячного триггера

Чтобы создать триггер, нужно выбрать его тип и указать все необходимые параметры, после чего нажать на кнопку “Ок” (рисунок 13). Если не все необходимые параметры были указаны, будет выведено соответствующие сообщение (рисунок 17). Для отмены создания необходимо нажать на кнопку “Отмена” (рисунок 13).

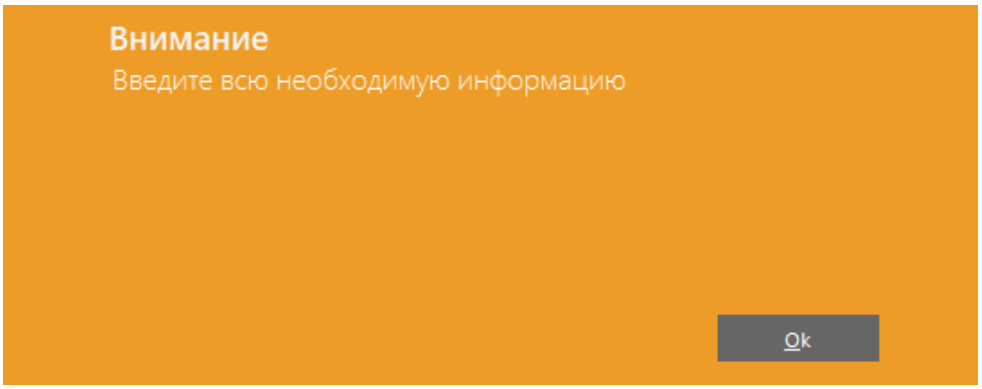


Рисунок 17 – Информационное сообщение

Чтобы удалить созданный триггер необходимо выбрать его в таблице и затем нажать на кнопку удалить “Удалить” (рисунок 12). Для изменения уже созданного триггера необходимо выбрать его в таблице и затем нажать на кнопку “Изменить” (рисунок 12).

Во вкладке “Безопасность” задается имя, и пароль пользователя от которого будут запущены действия задач (рисунок 18).

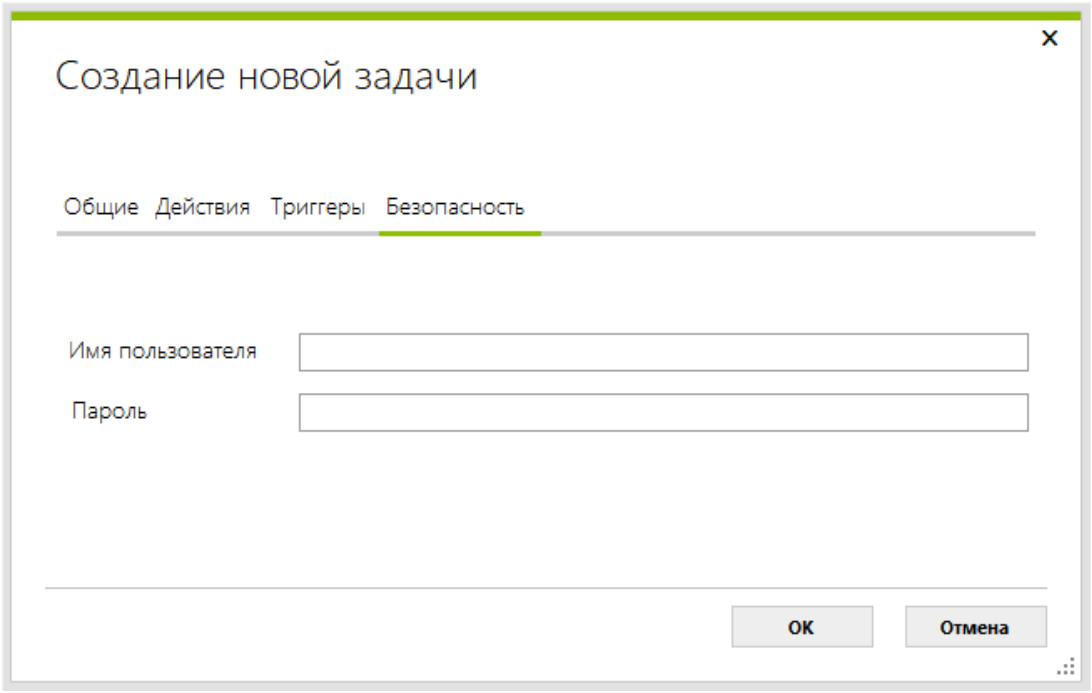


Рисунок 18 – Форма создания новой задачи. Вкладка “Безопасность”

Для создания задачи после указания всех её параметров необходимо нажать на кнопку “Ок’ (рисунок 10). Отменить создание задачи можно, нажав на кнопку “Отмена” (рисунок 10). Если имя создаваемой задачи уже занято будет выведено следующее информационное сообщение (рисунок 19).

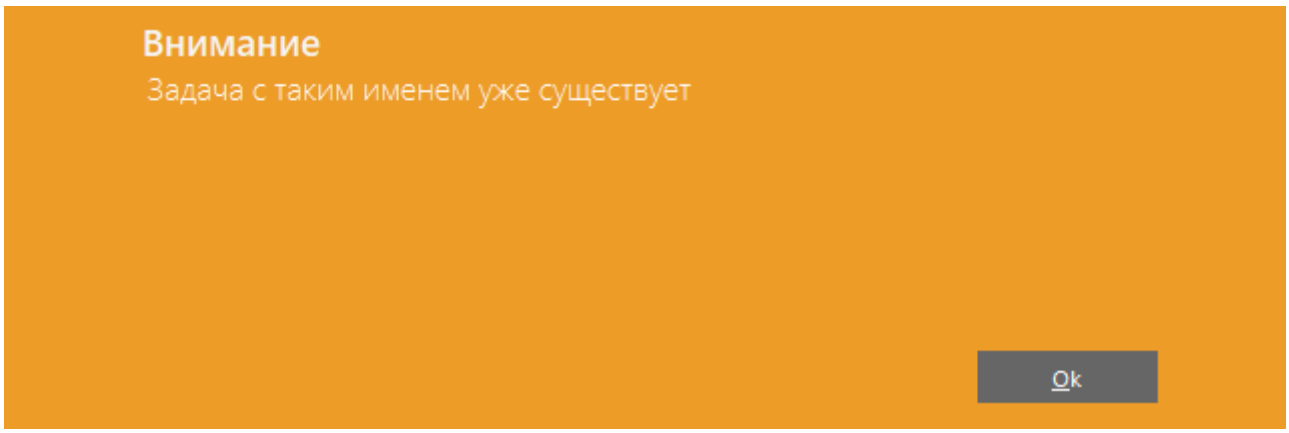


Рисунок 19 – Информационное сообщение

Для удаления созданной задачи необходимо выбрать её в таблице, и нажать на кнопку “Удалить” (рисунок 8). В случаи возникновения ошибки при удалении задачи, отображается соответствующее информационное сообщение (рисунок 20).

Рисунок 20 – Информационное сообщение

Для изменения созданной задачи необходимо выбрать её в таблице и нажать на кнопку “Изменить” (рисунок 8). Появится форма изменения задачи, визуально отличающаяся от формы создания задач только названием и цветом, но имеющая дополнительные проверки (рисунок 21). Все её поля будут заполнены значениями редактируемой задачи.

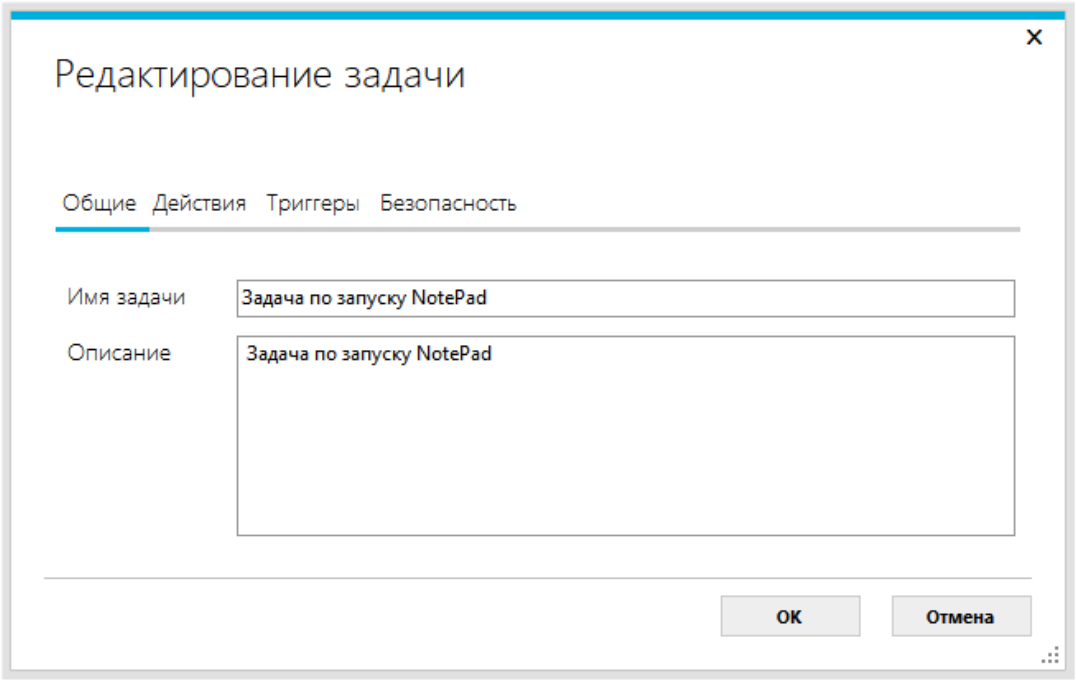


Рисунок 21 – Форма редактирования задачи

Для подключения к серверу необходимо нажать на кнопку “Соединение” (рисунок 8). В появившемся окне необходимо ввести ip-адрес или сетевое имя компьютера (рисунок 22).

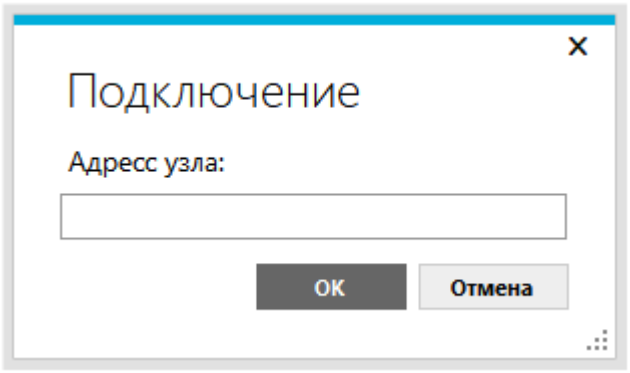


Рисунок 22 – Форма создания новой задачи. Вкладка “Безопасность”

Если сервер не установлен или не запущен на удаленном узле будет показано информационное сообщение (рисунок 23).

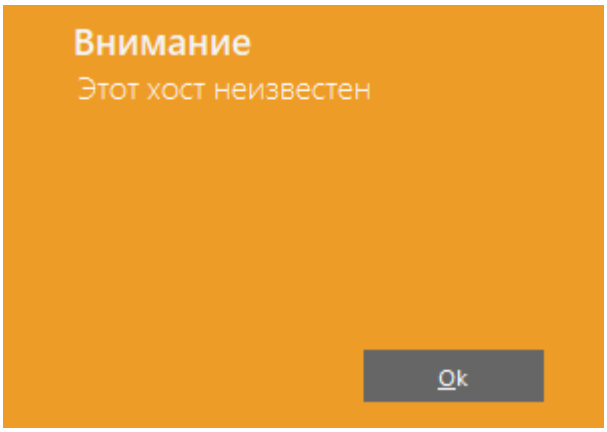


Рисунок 23 – Информационное сообщение

3.2 Спецификация сервера

Весь сервер разбит на отдельные модули, каждый из которых отвечает за свою функциональность, преимущественно все они разделены по разным файлам.

3.2.1 Модуль Task

Данный модуль содержит логику, связанную с задачами. В нем расположены структуры и функции для работы с ними. Рассмотрим некоторые из них.

STask – это структура, которая содержит всю информация, связанную с задачей. В ней хранится информация о действиях выполняемых данной задачей

и триггерах, инициирующих выполнение задачи. Здесь же располагаются данные о её последнем и ближайшем запусках. Вдобавок структура хранит сведения, связанные с пользователем, от лица которого будет запущено данная задач.

Основные функции для работы с задачами:

TaskCreate – эта функция, которая создает новую задачу. В ней происходит выделение необходимых задаче ресурсов и её инициализация.

TaskHashTableCreate – эта функция, которая создает хэш таблицу для хранения задач. Она выделяет память под хэш таблицу и инициализирует её.

TaskHashTableAddTask – эта функция, которая добавляет задачу в хэш таблицу.

TaskHashTableFindTask – эта функция, которая выполняет поиск задачи в хэш таблице. В случаи повторного добавления, возвращает ошибку.

TaskHashTableDeleteTask – эта функция, которая осуществляет удаление задачи из хэш таблицы.

TaskAdd – одна из наиболее важных функций, она производит действия по добавлению новой задачи во все модули системы. При этом выполняется добавление в глобальную хэш таблицу задач, сохранение в реестр и установка триггеров задачи.

TaskDelete– функция выполняет полное удаление задачи из всех модулей системы, а также из реестра.

TaskEdit – функция, изменяющая данные задачи.

TaskTriggersInstall – функция устанавливает все триггер из списка, располагающегося задаче.

TaskTriggersInstallAll – функция устанавливает все триггер из списка располагающийся в задаче.

TaskTriggersDeinstall – функция удаляет все триггеры для данной задачи.

TriggerInstall – функция устанавливает триггер.

TaskConvertToMessages – функция, разделяет задачу на массив сообщений, данная функция используется для передачи задач на клиента.

TaskActionsPerform – функция выполняющая все действия, связанные с задачей.

3.2.2 Модуль Action

Модуль содержит описание структуры содержащую информацию о действии. А также объявления и реализацию функция по управлению ими.

Основные функции для работы с задачами:

SAction – структура хранит информацию необходимую для нормального функционирования и запуска действий задач по расписанию. Отличительной особенностью является то, что в данной структуре содержится объединение. Данное объединение содержит структуры, содержащие информацию характерную для различных типов действий. Конкретный тип действия, хранящийся в объединении в данный момент, определяется одним их полей SAction. Такой подход хранения данных позволяет избегать создания различных структур под разные действия, дублируя поля в них.

SActionExecuteProgramm – структура, является одним из полей объединения расположенного в SAction. В ней содержится информация о программе или скрипте, который будет исполнен.

Основные функции для работы с действиями:

ActionCreate – функция создания нового действия.

ActionAddToList – функция добавления действия в список действий.

ActionListInitialize – функция инициализации списка действий.

3.2.3 Модуль Trigger

Модуль содержит описание структур с данными о различных типах триггерах и функций для работы с ними. Как и структура действия, содержит объединение для хранения данных различных триггеров.

STrigger – структура, содержит данные характерные для всех видов триггеров.

ETriggerType – это перечисление содержит типы триггеров.

ETimeTriggerType – перечисление, содержит типы временных триггеров.

STriggerTime – структура, содержит информацию характерную для всех временных триггеров.

STriggerList – структура, представляет собой список триггеров.

Основные функции для работы с триггерами:

TriggerCreate – функция, которая выполняет создание нового триггера.

TriggerListInitialze – функция инициализации списка триггеров.

TriggerListAddTrigger – функция добавления триггера в список триггеров.

3.2.4 Модуль Registry

В данном модуле содержится набор функций для работы с реестром. Они предназначены для сохранения и изъятия информации об основных структурах, данные которых должны сохраняться для долговременного хранения.

SaveTaskToRegistry – функция, которая сохраняет задачу в реестр. При этом сохраняются все действия и триггеры, относящиеся к ней.

SaveTriggersToRegistry – функция для сохранения информации о триггере в реестр.

SaveActionToRegistry – функция, сохраняющая информацию о действии в реестр.

DeleteTaskFromRegistry – функция, которая удаляет информацию из реестра связанную с указанной задачей.

ReadTaskFromRegistry – функция предназначена для изъятия из реестра задачи, в том числе всех триггеров и действий связанные с ней.

GetTasksFromRegistry – функция осуществляющая изъятие всех задач из реестра.

ReadActionFromRegistry – функция считывания одного действия задачи из реестра.

ReadTriggerFromRegistry – функция считывания одного триггера задачи из реестра.

ReadActionsFromRegistry – функция считывания списка действий задачи из реестра.

ReadTriggersFromRegistry – функция считывания списка триггеров задачи из реестра.

3.2.5 Модуль Service

Рассматриваемый модуль содержит функции, отвечающие за работу службы. Здесь осуществляется прием и обработка запросов от диспетчера управления служб.

ServiceMain – главная функция службы. В ней осуществляется начальная настройка и запуск рабочих нитей. Настройка включает в себя считывание списка задач из реестра и установка их триггеров.

ServiceCtrlHandler – обработчик событий от диспетчера управления служб. Здесь устанавливается событие завершения работы службы.

InstallService – функция установки службы в диспетчере управления службами.

LaunchService – функция запуска службы.

RemoveService – функция удаления службы.

3.2.5 Модуль RequestHandler

Модуль осуществляет прием, обработку и выполнение запросов, поступающих из клиентской программы. Из этого следует, что данный модуль является одним из главных. При этом данный модуль требует синхронизации, так как работает в отдельной нити.

В процессе работы нить большую часть времени находится в спящем состоянии, в связи с ожиданием наступления события. Запрос от клиента, пробуждает нить. Затем она выставляет блокировку для модулей, работающих в других нитях, и начинает обработку.

Основные структуры и функции, используемые в этом модуле:

SReqResInfo – структура хранит информацию о запросе такую как количество сообщений и тип запроса. Также используется при посылке ответа клиенту.

SReqResData – структура, содержит данные передаваемые в запросах и ответах.

EReqResDataType – перечисление, содержит набор именованных констант, определяющих типы данных, содержащиеся в запросах и ответах.

EReqResInfoType – перечисление, содержит набор именованных констант, определяющих различные виды запросов и ответов.

ConnectToNewClient – функция, осуществляет подключение к клиенту. Не блокирует выполнение нити.

ParseMessages – функция преобразования сообщений в задачу.

GetPipeInstance – функция, возвращает экземпляр именованного канала. К созданному каналу подключается клиент.

SendErrorToClient – функция посылки сообщения с ошибкой клиенту.

PerformRequest – функция осуществляет выполнение запроса, поступившего от клиента. При работе осуществляется синхронизация работы с остальными нитями.

3.2.6 Модуль Message

Рассматриваемый модуль носит вспомогательный характер. Функции и структуры данного модуля необходимы для централизованного хранения сущностей различных типов данных. Используется при отправке и приеме данных при общении с клиентом.

Основные структуры и функции, используемые в этом модуле:

SMessage – структура, содержит данные и их размер, хранящиеся в сообщении.

SMessageArray – структура для хранения сообщений. Является массивом со счетчиком количества сообщений.

MessageCreate – функция создания нового сообщения. Выделяет память под сообщение и инициализирует поля начальными значениями.

MessageFree – функция, которая выполняет корректную очистку ресурсов сообщения.

MessageArrayCreate – функция создания списка сообщений. Производит предварительное выделение памяти для нескольких сообщений.

MessageArrayAddMessage – функция добавления нового сообщения в массив.

MessageArrayDeleteMessage – функции удаление сообщение из списка. Удаление осуществляется по адресу сообщения.

MessageArrayFreeAllMessages – функция производит освобождение памяти всех сообщений в массиве.

MessageArrayFree – функция, освобождающая ресурсы массива сообщений. Не освобождает ресурсы хранящихся в ней сообщений.

3.2.7 Модуль Security

Модуль выполняет действия по обеспечению безопасности при выполнении задачи. Задачи могут быть созданы разными пользователями имеющие разные права. Здесь обеспечивается запуск задачи с прав доступа указанного пользователя.

Основные структуры и функции, используемые в этом модуле:

SSecurity – структура содержит информацию о пользователе. Данные из неё используются при запуске задач.

SecurityGetUserToken – функция определяет пользователя от лица которого будут запущена задача и возвращает его маркер.

3.2.8 Модуль Time

В этом модуле реализована нить ожидающая срабатывание временных триггеров. Для обеспечения корректной работы сервера модуль синхронизирован с другими нитями. Все операции по изменению триггеров и их таймеров требуют предварительной приостановки их ожидание. Кроме того, в нём объявлен набор вспомогательных функций для работы со временем.

Основные структуры и функции, используемые в этом модуле:

STimersContext – структура хранит информацию об установленном для каждого триггера таймере. Доступ на изменение к этой структуре синхронизирован между нитями.

STimerDetails – структура, содержит данные связанные с таймером.

TimerDetailsCreate – функция создания структуры STimerDetails.

TimersContextCreate – функция создания структуры STimersContext.

TimerContextAddTimer – функция добавления нового таймера.

TimerContextDeleteTimer – функция удаления таймера.

TimerContextGetTimerIndexByTask – функция получения индекса таймера, связанного с триггером переданной в неё задачи.

WaitTimersLock – функция останавливает ожидание таймеров. Используется с целью обеспечения

WaitTimersUnLock – функция возобновляет ожидания таймеров.

WaitForTimersThread – функция нити, реализующая ожидания наступления времени, установленного в таймерах. В ней осуществляется процесс запуска действий задачи и переустановку таймеров на новое время.

Основные функции для работы со временем:

GetDailyTriggerTime – функция определения времени срабатывания ежедневного триггера, на основе в его начального времени и частоты запуска.

GetWeeklyTriggerTime – функция определяет время срабатывания еженедельного триггера, на основе его начального времени и дней недели.

GetDailyMonthTriggerTime – функция рассчитывает время срабатывания ежемесячного триггера, на основе указанных месяцев и дней в них.

GetWeeklyMonthTrigger – функция рассчитывает время срабатывания ежемесячного триггера, на основе указанных месяцев, дней недели, а также недель в месяце.

GetfTimesDiffInDays – функция определяет разницу между двумя датами в днях. Является вспомогательной для других функций, работающих со временем.

DateToDayOfWeek – функция рассчитывает день недели, переданной в неё даты.

3.2.9 Модуль GenericContainers

Данный модуль содержит реализацию различных структур данный таких как массив, список, хэш-таблицы и связанная хэш таблица. Они предназначены для организации эффективного управления данными. Структуры не предполагают хранение данных определенного типа. Следовательно, могут быть повторно использованы для хранения различных структур, что уменьшает дублирование кода.

Основные структуры и функции для работы со всеми контейнерами:

SGenericElement – структура предназначена расположения в контейнерах. Применяется для инкапсуляции данных различных типов.

EElementType – перечисление, содержит набор именованных констант, определяющих типы данных в контейнерах.

GenericElementCreate – функция создания экземпляра структуры SGenericElement.

GenericElementFree – функция освобождения занятых ресурсов.

Структуры и функции для работы с массивом:

SGenericArray – структура представляет собой массив со счетчиком элементов в нем.

GenericArrayCreate – функция создания экземпляра массива.

GenericArrayAddElement – функция добавления элемента в массив.

GenericArrayDeleteElement – функция удаления элемента из массива.

GenericArrayGetIndexOfElement – функция получения индекса элемента хранящегося в массиве.

GenericArrayGetElementByIndex – функция получения элемента массива по его индексу.

Структуры и функции для работы со списком:

SGenericEntry – структура для хранения элемента двусвязного списка.

SGenericList – структура представляет собой двусвязный список. При этом может быть может работать как множество, режим работы указывается при создании.

GenericListCreate – функции создает экземпляр списка, при этом инициализируя её поля значениями по умолчанию.

GenericListAddElement – функция добавления элемента в список. Добавление осуществляется в конец списка.

GenericListDeleteElement – функция удаления элемента из списка. Перед удалением производится поиск элемента по адресу.

GenericListFindElement – функция поиск элемента в списке.

GenericListFree – функция освобождения ресурсов, занятые списком.

Структуры и функции для работы с хэш-таблицей:

SHashTable – структура представляется собой хэш-таблицу. В её основе лежит массив списков, описанные выше.

GetHash – функция расчета хэш-значения. Функция принимает данные различных типов и длин.

HashTableCreate – функция создания экземпляра хэш-таблица.

HashTableAddElement – функция добавления элемента в хэш-таблицы.

HashTableDeleteElement – функция удаления элементов из хэш-таблицы.

HashTableFindTable – функция для поиска элемента в хэш-таблице.

HashTableFree – функция освобождения ресурсов занятые хэш-таблицей.

Для увеличения эффективности при перечислении элементов хэш-таблицы, на её основе реализована связанная хэш-таблица. Именно эта структура данных используется в программе для хранения задач.

Структуры и функции для работы со связанной хэш-таблицей:

SLinkedHashTable – это структура, которая используется в качестве связанной хэш-таблицы.

SLinkedHashTableEntry – структура обеспечивающая связь между элементами.

LinkedHashTableCreate – функция для создания экземпляра связанной хэш-таблицы. Обеспечивает корректное выделение ресурсов, и начальную инициализацию.

LinkedHashTableAddElement – функция вставки элемента в связанную хэш-таблицу. В ней происходит связь элементов.

LinkedTableDeleteElememt – функция удаления элемента из хэш-таблицы.

LinkedTableFindElement – функция поиска элемента в связанной хэш-таблице.

LinkedHashTableFree – функция освобождения ресурсов занятых связанной хэш-таблицей.

LinkedHashTableGetFirstEntry – функция получения первой записи из связанной хэш-таблицы. Используется при перечислении элементов.

LinkedHashTableGetNextEntry – функция получения следующей записи в связанной хэш-таблицы. Используется при перечислении элементов.

3.3 Спецификация клиента

Клиент разработан с использованием шаблона проектирования MVC. Язык программирования C#, на котором реализован клиент, позволил полностью реализовать все концепции шаблона MVC.

3.3.1 Классы модели

Классы, формирующие модель, предоставляют данные и методы для работы с ними. Они не зависят от представления и контроллера. Следовательно, не представляют средств визуализации данных и не обеспечивают взаимодействие с пользователем. Таким образом на их основе можно строить различные виды представлений, и обеспечивать несколько интерфейсов взаимодействия, не меняя при этом код модели.

Task – главный класс модели. Он хранит все параметры задачи, действия выполняемый ей, а также триггеры, запускающие задачу. Методы выполняют запросы к службе для добавления, удаления и изменения задачи, являющейся экземпляром данного класса.

Основные методы и поля класса Task:

name – поле, хранит имя задачи в строковом представлении.

description – поле, хранит описание задачи в строковом представлении.

security – поле типа Security, в нем хранится информация о пользователе с парами которого запускаются действия задачи.

lastExecute – это поле хранит время последнего запуска задачи.

nextExecute – это поле хранит время следующего запуска задачи.

triggerList – список триггеров задачи, инициирующие выполнение задачи.

actionList – список действий, выполняемых задачей при запуске.

ConvertToMessage – метод преобразования задачи в список сообщений. Он используется для передачи данных задачи на сервер. В процессе преобразованию подвергаются все действия и триггеры данной задачи.

Add – метод для добавления задачи на сервер. Применятся после формирования задачи.

Delete – метод производит удаление задачи на сервере. В качестве идентификатора задачи для удаления используется её имя.

GetAll – метод для получения списка задач от сервера.

Trigger – абстрактный класс, является базовым для всех триггеров. Используется в качестве типа элементов обобщенных контейнеров при хранении различных триггеров.

Основные методы и поля класса Trigger:

workTime – поле определяет даты окончания работы триггера. После её наступления триггер деактивируется.

isWorkTimeEnable – поле, является флагом учета worktime при работе триггера. Если флаг не выставлен триггер никогда не будет деактивирован.

ConvertToMessage – метод для преобразования триггера в сообщение. Данный метод должен быть реализован всеми потомками данного класса.

TimeTrigger – абстрактный класс, потомком класса Trigger, является базовым для всех временных триггеров. Основное отличие – это указываемое время срабатывания данного триггера.

OnceTrigger – класс однократного триггера, является потомком класса TimeTrigger. Ничего не добавляет к базовому, необходим для создания экземпляров базового класса. Представляет собой триггер, срабатывающий однократно в указанное время.

DailyTrigger – класс ежедневного триггера, является потомком класса TimeTrigger. Представляет собой триггер, срабатывающий с частотой, указываемой в днях в установленное время.

WeeklyTrigger– класс еженедельного триггера, является потомком класса TimeTrigger. Условиями срабатывания триггера являются: наступление указанного времени, определенный день недели.

MonthyTrigger– абстрактный класс ежемесячного триггера, является потомком класса TimeTrigger. На его основе создаются более сложные вариации триггеров.

DailyMonthTrigger – класс ежемесячного-дневного триггера, является потомком класса MonthyTrigger. В нем указываются не только месяц, но и дни его срабатывания.

WeeklyMonthTrigger – класс ежемесячного-недельного триггера, является потомком класса MonthyTrigger. Это комплексный триггер, учитывающий множество условий таких как: месяц, день недели, порядок недель в месяцах.

Вспомогательные структуры, используемые классами триггеров:

SDaysOfWeek – структура определяющая активность дней в недели.

SMonths – структура определяющая активность месяцев в году.

Action– абстрактный класс, является базовым для всех типов действий. Используется в качестве типа элементов обобщенных контейнеров при хранении различных действий.

Основные методы и поля класса Action:

isEnable – поле, являющееся флагом активности действия. Деактивированные действия не запускаются при выполнении задачи.

ConvertToMessage – метод преобразования задачи в сообщение. Используется при передаче задачи на сервер. Данный метод должен быть переопределен всеми потомками данного класса.

ProgrammExecuteAction – класс действия, запускающего программу. Является потомком класса Action. Предназначен для запуска программы или скрипта.

Security – класс хранения информацию о пользователе с правами которого запускаются действия задач.

ServerWorker – класс обеспечивает взаимодействие клиента с сервером. В нём реализованы методы для отправки запросов и принятия ответа, а также вспомогательные для преобразования структура в массив байт.

Основные методы и поля класса ServerWorker:

SendAcyncRequestToService – метод асинхронной отправки запроса на сервер.

SendRequestToService – метод отправляет запрос на сервер синхронно. После отправки ожидает ответа от сервера.

3.3.2 Класс контроллера

Контроллерами являются классы, обеспечивающие связь между системой и пользователем. Используя представление и модель, контроллер реализует различные реакции в ответ на воздействия пользователей.

TaskController – класс контроллер. В нем находятся методы, используемые различными представлениями для реализации одних и тех же действий.

Основные методы и поля класса TaskController:

listOfTasks – поле, являющееся списком задач. Данный список формируется при старте программы и изменяется в процессе её работы.

selectedTask – поле, является задачей выбранной в данный момент. Практически все действия нацелены на выбранную задачу.

RefreshTasks – событие обновления списка задач. Данное события инициируется самим контроллером для уведомления всех подписавшихся представлений об обновлении списка задач.

SelectedTaskChanged – метод вызываемый при выборе новой задачи. Необходим для корректной обработки выбора задачи и уведомлении об это событии.

AddTask – метод реализует добавление задачи. Используется представлениями в качестве интерфейса для изменения модели.

DeleteTask – метод реализует удаление задачи. Используется представлениями в качестве интерфейса для изменения модели.

Refresh – метод обновления информации о всех задачах. Получает список задач с сервера и заменяет старые на новые.

EditTask – метод изменения существующей задачи. На основе уже имеющейся задачи создает новую с указанными параметрами и заменят старую версию.

CreateNewTask – метод создания новой задачи. В процессе выполняются различные вариационные действия её параметров. В том числе, проверятся уникальности имени задачи.

3.3.3 Классы представления

Классы представления отображают конкретное состояние модели и передают воздействия пользователей посредством контроллера. Клиент использует различные представления для отображения моделей описанных выше.

ActionView – класс представления, используется для отображения действия. Данный класс по средству композиции включает ряд других классов расширяющие его для возможности представления различных типов действий.

Основные методы и поля класса ActionView:

ShowAction – метод, отображающие переданное в него действие задачи.

GetAction – метод для получения действия задачи, созданного из данных находящихся в представлении.

Verify – метод, который проверяет приемлемость параметров действия, заданные в представлении.

IActionOption – интерфейс, реализуемый всеми представлениями отображающие параметры различных действий.

GetActionOptions – метод описанный в интерфейсе IActionOption. Возвращает действие созданное на основе данных из представления.

ExecuteProgramAction – класс представления, который содержащий действие запуска задачи. Данный класс реализует интерфейс IActionOption, а также используется в классе ActionView.

TriggerView – класс представления, используется для отображения триггера. Данный класс по средству композиции включает ряд других классов расширяющие его для возможности представления различных типов триггеров.

Основные методы и поля класса TriggerView:

activeParam – поле класса TriggerView содержит текущий набор параметров триггера.

TriggerParams – поле класса TriggerView, является словарем из всех наборов параметров различных триггеров.

ShowTrigger – метод осуществляющий показ триггера, переданного в представление.

GetTrigger – метод формирования триггер на основе сведений из представления.

Verify – метод класс TriggerView. Проверяет приемлемость параметров триггера из представления.

ITriggerOptions – интерфейс, реализуемый всеми представлениями отображающие параметры различных триггеров.

GetTriggerOptions – метод описанный в интерфейсе ITriggerOptions. Возвращает триггер созданный на основе данных из представления.

TimeTrigger – класс представления, содержащий параметры характерные для всех временных триггеров. Данный класс реализует интерфейс ITriggerOptions, а также используется в классе TriggerView.

DailyTrigger – класс представления, содержащий параметры ежедневного триггера. Данный класс реализует интерфейс ITriggerOptions, а также используется в классе TriggerView.

WeeklyTrigger – класс представления, содержащий параметры еженедельного триггера. Данный класс реализует интерфейс ITriggerOptions, а также используется в классе TriggerView.

MonthTrigger – класс представления, содержащий параметры ежемесячного триггера. Данный класс реализует интерфейс ITriggerOptions, а также используется в классе TriggerView.

CustomTable – обобщенный класс представления. Предназначен для отображения списков различных типов данных. Используется для отображения списков триггеров и действий задач.

ProgressDialog – класс представления. Отображает прогресс выполнения задачи.

3.3.4 Классы форм

MainForm – класс главной формы программы. Она отображается при запуске клиента. В ней расположена таблица с задачами и кнопки для осуществления различных действий над ними.

TaskOptionsForm – класс формы содержащий основные настройки задачи. Предназначен для редактирования имеющихся и создания новых задач.

ConnectionForm – класс формы предназначенный для подключения к серверу. Пользователь может ввести либо ip-адрес, либо имя компьютера в сети.

ActionOptionForm – класс формы содержащий основные настройки действия. Предназначен для редактирования имеющихся и создания новых действий.

TriggerOptionForm – это класс формы содержащий основные настройки триггера. Предназначен для редактирования имеющихся и создания новых триггеров.

3.3.5 Интерфейс с севером

Для обеспечения связи между данными расположенными на сервере и клиенте, создан набор структур и перечислений обеспечивающий его. Так как сервер написан на языке Си (не управляемый код), а клиент на языке C# (управляемый код) для обеспечения соответствия данных используется маршлинг.

SAction – структура действия. Полностью соответствует структуре, расположенной на сервере.

SRequstData – структура содержащая данные передаваемые в запросе к серверу и ответах от него.

SRequestInfo – структура содержит мета данные о запросе и ответе.

UnmanagedSecurity – структура содержит данные о пользователе.

UnmanagedTaskInfo – структура содержит данные о задаче.

STrigger – структура содержит данные о триггере. Полностью соответствует структуре, расположенной на сервере.

UnmanagedTimeTrigger – структура содержит данные о временных триггерах

Заключение

Результатом данной работы является клиент-серверное приложение “Планировщик задач”. Приложение позволяет создавать задачи со сложными, гибкими условиями их выполнения, используя различные типы триггеров. Задачи могут выполнять несколько действий, исполняемых как в фоновом, так и в интерактивном режимах. Пользователь может получать задачи с удаленных серверов или сервера, установленного на локальной машине. С помощью разработанного приложения можно создавать автономно работающие системы и удаленно управлять их работой. Оно полностью соответствует всем заявленным требованиям.

В процессе тестирования приложения были выявлены различные проблемы. Клиент мог создавать множество задач с одинаковыми именами. Данная проблема была устранена путем использования хеш-таблицы на стороне сервера, ключами в которой выступали имена задач. Сервер портил данные. Ошибка была устранена с помощью захвата блокировки для доступа к разделяемым ресурсам.

В дальнейшем приложение может быть улучшено путем добавления новых типов триггеров и действий. Например, триггером может являться поступивший запрос на сервер, или определенные значения показателей аппаратуры компьютера. Реализация связей между задачами также увеличит гибкость приложения. С помощью них станет возможно создавать рекурсивные взаимосвязи. На стороне клиента можно добавить разделения задач по разделам, что позволит более структурированно управлять ими. Также возможно добавить функцию принудительного исполнения задачи, вне зависимости от состояния её триггеров.

В ходе выполнения работы, были получены навыки разработки клиент-серверного приложения. Усовершенствованы умение выполнять декомпозицию предметной области и выделять в ней основные сущности. Значительно были

развиты навыки применения основных принципов объектно-ориентированного программирования, такие как наследование и полиморфизм. Был получен опыт в создании протокола взаимодействия клиента и сервера, а также в использовании шаблона MVC.

Список используемых источников

1. MSDN – URL: https://msdn.microsoft.com/
2. Руссинович М. и Соломон Д. «Внутреннее устройство Microsoft Windows». 2013 г, стр. 800.
3. Эрих Гамма, Ричард Хелм, Ральф Джонсон, Джон Влиссидес «Приемы объектно-ориентированного проектирования. Паттерны проектирования». 2016 г., 366 стр.
4. Джеффри Рихтер «CLR via C#. Программирование на платформе Microsoft.NET Framework 4.5 на языке C#». 2017 г., 896 стр.
5. Билл Вагнер «Наиболее эффективное программирование на C#. 50 способов улучшения кода». 2017 г., 224 стр.
6. https://www.helpsystems.com/resources/guides/enterprise-job-scheduling-how-your-organization-can-benefit-automation

Приложение А

Листинги программы клиента

Приложение А.1

Файл «MainForm.cs»

```
namespace TaskEditor
{
    public partial class MainForm : MetroFramework.Forms.MetroForm
    {
        TaskViewer.Controllers.TaskController taskController;

        private BindingSource tasksBindingSource = new
BindingSource();

        public MainForm()
        {
            InitializeComponent();

            taskController = new
TaskViewer.Controllers.TaskController();
            taskController.RefreshTasks +=
TaskController_RefreshTasks;

            try
            {
                taskController.GetTasks();
            }
            catch (Exception err)
            {
                ShowMessage(err.Message, "Ошибка",
MessageBoxIcon.Error);
            }
            this.tableOfTasks.AutoSizeColumnsMode =
DataGridViewAutoSizeColumnsMode.Fill;
```

```
            this.tableOfTasks.MultiSelect = false;

            tableOfTasks.RowStateChanged +=
taskController.SelectedTaskChanged;
        }

        private void
TaskController_RefreshTasks(List<TaskEditor.Task> listOfTask)
        {
            this.tableOfTasks.DataSource = null;

            this.tableOfTasks.DataSource = listOfTask;
        }

        private void btnCreateTask_Click(object sender, EventArgs
e)
        {
            try
            {
                taskController.CreateNewTask();
            }
            catch (WarningException err)
            {
                ShowMessage(err.Message, "Внимание",
MessageBoxIcon.Warning);
            }
            catch (Exception err)
            {
                ShowMessage(err.Message, "Ошибка",
MessageBoxIcon.Error);

            }
        }

        public void Clear()
        {
```

```
            this.tableOfTasks.DataSource = null;
        }

        private void ShowMessage(String message, String title,
MessageBoxIcon type)
        {
            MetroFramework.MetroMessageBox.Show(this, message,
title, MessageBoxButtons.OK, type);
        }

        private void btnDeleteTask_Click(object sender, EventArgs
e)
        {
            try
            {
                taskController.DeleteTask();
            }
            catch (WarningException err)
            {
                ShowMessage(err.Message, "Внимание",
MessageBoxIcon.Warning);
            }
            catch (Exception err)
            {
                Clear();
                ShowMessage(err.Message, "Ошибка",
MessageBoxIcon.Error);
            }
        }

        private void btnEditTask_Click(object sender, EventArgs e)
        {
            try
            {
                taskController.EditTask();
            }
```

```
            catch (WarningException err)
            {
                ShowMessage(err.Message, "Внимание",
MessageBoxIcon.Warning);
            }
            catch (Exception err)
            {
                Clear();
                ShowMessage(err.Message, "Ошибка",
MessageBoxIcon.Error);
            }
        }

        private void metroTile1_Click(object sender, EventArgs e)
        {
            try
            {
                taskController.Refresh();
            }
            catch (WarningException err)
            {
                ShowMessage(err.Message, "Внимание",
MessageBoxIcon.Warning);
            }
            catch (Exception err)
            {
                Clear();
                ShowMessage(err.Message, "Ошибка",
MessageBoxIcon.Error);
            }
        }

        private void btnConnect_Click(object sender, EventArgs e)
        {
            try
            {
```

```
                TaskViewer.Forms.ConnectionForm connection =
                    new TaskViewer.Forms.ConnectionForm();

                if (connection.ShowDialog() == DialogResult.OK)
                {

                    Service.hostIp =
(connection.hostEntry.AddressList.Length > 0 ) ?

connection.hostEntry.AddressList[0].ToString() :
connection.hostEntry.HostName;

                    taskController.Refresh();
                }

            }
            catch (WarningException err)
            {
                ShowMessage(err.Message, "Внимание",
MessageBoxIcon.Warning);
            }
            catch (Exception err)
            {
                ShowMessage(err.Message, "Ошибка",
MessageBoxIcon.Error);
            }
        }
    }
}
```

Приложение А.2

Файл «TaskController.cs»

```
namespace TaskViewer.Controllers
```

```
{
    public class TaskController
    {
        /* Список все задач*/
        private List<TaskEditor.Task> listOfTasks;

        private TaskEditor.Task selectedTask = null;

        public delegate void TaskListHandler(List<TaskEditor.Task>
listOfTask);

        public delegate void TaskHandler(TaskEditor.Task task);

        public event TaskListHandler RefreshTasks;

        public TaskController()
        {
            if (listOfTasks == null)
                listOfTasks = new List<TaskEditor.Task>();
        }

        public void SelectedTaskChanged(object sender,
DataGridViewRowStateChangedEventArgs e)
        {
            TaskEditor.Task task;
            if (e.StateChanged !=
DataGridViewElementStates.Selected) return;
            try
            {
                task = e.Row.DataBoundItem as TaskEditor.Task;

            }
            catch
            {
                return;
            }
```

```
        selectedTask = task;
    }

    public void AddTask(TaskEditor.Task newTask)
    {
        newTask.Add();

        listOfTasks.Add(newTask);

        RefreshTasks.Invoke(listOfTasks);
    }

    public void RemoveTask(TaskEditor.Task task)
    {
        task.Delete();

        listOfTasks.Remove(task);

        RefreshTasks.Invoke(listOfTasks);
    }

    public void DeleteTask()
    {
        if (selectedTask != null)
        {
            selectedTask.Delete();

            listOfTasks.Remove(selectedTask);

            selectedTask = null;

            RefreshTasks.Invoke(listOfTasks);
        }
    }
```

```
        public void GetTasks()
        {
            /* Получение списка задач из службы */
            try
            {
                listOfTasks = TaskEditor.Task.GetAll();

                RefreshTasks.Invoke(listOfTasks);

            }
            catch (Exception err)
            {
                listOfTasks.Clear();
                RefreshTasks.Invoke(listOfTasks);

                throw err;
            }
        }

        public void Refresh()
        {
            listOfTasks.Clear();

            GetTasks();
        }

        public void CreateNewTask()
        {

            TaskViewer.Forms.TaskOptionForm createTaskForm =
                new
TaskViewer.Forms.TaskOptionForm(MetroFramework.MetroColorStyle.Lim
e,
                listOfTasks,
                Forms.TaskOptionType.TASK_CREATE,
                selectedTask
```

```
                );

            switch (createTaskForm.ShowDialog())
            {
                case System.Windows.Forms.DialogResult.OK:
                    TaskEditor.Task newTask =
createTaskForm.GetTask();

                    AddTask(newTask);

                    break;
                case System.Windows.Forms.DialogResult.Cancel:

                default:
                    break;
            }
        }

        public void EditTask()
        {
            if (selectedTask == null)
                throw new WarningException("Задача не выбрана");

            TaskViewer.Forms.TaskOptionForm editTaskForm =
                 new
TaskViewer.Forms.TaskOptionForm(MetroFramework.MetroColorStyle.Blu
e,
                 listOfTasks,
                Forms.TaskOptionType.TASK_EDIT,
                selectedTask);

            editTaskForm.Text = "Редактирование задачи";

            editTaskForm.ShowTask(selectedTask);

            switch (editTaskForm.ShowDialog())
```

```
            {

                case System.Windows.Forms.DialogResult.OK:

                    TaskEditor.Task newTask =
editTaskForm.GetTask();

                    newTask.lastExecute =
selectedTask.lastExecute;

                    RemoveTask(selectedTask);

                    AddTask(newTask);

                    break;
                case System.Windows.Forms.DialogResult.Cancel:

                default:
                    break;
            }
        }
    }
}
```

Приложение А.3

Файл «TriggerView.cs»

```
namespace TaskViewer.Views.Trigger
{
    public partial class TriggerView : UserControl
    {
        private Dictionary<string, ITriggerOptions> TriggerParams;

        private KeyValuePair<string, ITriggerOptions> activeParam;
```

```
        private MetroFramework.MetroColorStyle style =
            MetroFramework.MetroColorStyle.Default;

        public MetroFramework.MetroColorStyle Style
        {
            set
            {
                style = value;
                cbEnable.Style = style;
                cbWorkTime.Style = style;
                combTriggerType.Style = style;
                timeTrigger.Style = style;
            }
        }


        public TriggerView()
        {
            InitializeComponent();


            /* При изменении типа триггера изменяются параметры */
            combTriggerType.SelectedValueChanged +=
ShowTriggerParam;

            /* Типы триггеров ассоциированные с панелью параметров
*/
            TriggerParams = new Dictionary<string,
ITriggerOptions>();

            /* Инициализация типов триггеров и панелей с
параметрами */
            TriggerParams.Add("По расписанию", this.timeTrigger);
            TriggerParams.Add("При простои ",
(ITriggerOptions)noParamTrigger);
```

```
            TriggerParams.Add("При входе в систему ",
(ITriggerOptions)noParamTrigger);
            TriggerParams.Add("При блокировки",
(ITriggerOptions)noParamTrigger);


            /* Заполнение типов триггеров */
            foreach (KeyValuePair<string, ITriggerOptions>
triggerParam in TriggerParams)
            {
                combTriggerType.Items.Add(triggerParam.Key);
            }

            activeParam = new KeyValuePair<string,
ITriggerOptions>
                ("По расписанию", TriggerParams["По расписанию"]);

            /* Выбрать тип триггера */
            combTriggerType.SelectedItem = activeParam.Key;
        }


        public void ShowTrigger(TaskEditor.Trigger trigger)
        {
            cbEnable.Checked = trigger.isEnable;

            cbWorkTime.Checked = trigger.isWorkTimeEnable;

            dateWork.Value = trigger.workTime;
            timeWork.Value = trigger.workTime;

            switch (trigger.triggerType)
            {
                case TaskEditor.ETriggerType.TRIGGER_TIME:
```

```
                    combTriggerType.SelectedText = "По
расписанию";


                    activeParam =  new KeyValuePair<string,
ITriggerOptions>
                ("По расписанию", TriggerParams["По расписанию"]);


((TriggerTypeOption.TimeTrigger)TriggerParams["По расписанию"]).

ShowTrigger((TaskEditor.TimeTrigger)trigger);

                    break;
                case TaskEditor.ETriggerType.TRIGGER_LOGON:
                    break;
                case TaskEditor.ETriggerType.TRIGGER_IDLE:
                    break;
                case TaskEditor.ETriggerType.TRIGGER_SYSTEMSTART:
                    break;
                case TaskEditor.ETriggerType.TRIGGER_LOGOUT:
                    break;
                case TaskEditor.ETriggerType.TRIGGER_ENABLE:
                    break;
                default:
                    break;
            }
        }

        public TaskEditor.Trigger GetTrigger()
        {
            TaskEditor.Trigger trigger = null;

            if (activeParam.Key == "По расписанию")
            {
                trigger = activeParam.Value.GetTriggerOptions();
```

```
            }

            trigger.isEnable = cbEnable.Checked;

            trigger.workTime = this.timeWork.Value;

            trigger.workTime = new DateTime(
                this.timeWork.Value.Year,
                this.timeWork.Value.Month,
                this.timeWork.Value.Day,
                this.timeWork.Value.Hour,
                this.timeWork.Value.Minute,
                this.timeWork.Value.Second);

            trigger.isWorkTimeEnable = cbWorkTime.Checked;


            return trigger;
        }


        void ShowTriggerParam(object sender, EventArgs e)
        {
            MetroFramework.Controls.MetroComboBox combo = sender
as MetroFramework.Controls.MetroComboBox;

            if (activeParam.Value != null)
                ((Control)activeParam.Value).Visible = false;

            activeParam = new KeyValuePair<string,
ITriggerOptions>
                  ((String)combo.SelectedItem,
TriggerParams[(String)combo.SelectedItem]);

            ((Control)activeParam.Value).Visible = true;
        }
```

```
        public bool Verify()
        {
            if (activeParam.Key == "По расписанию")
            {
                return activeParam.Value.Verify();
            }

            return false;
        }
    }
}
```

Приложение А.4

Файл «TimeTrigger.cs»

```
namespace TaskViewer.Views.Trigger.TriggerTypeOption
{
    public partial class TimeTrigger : UserControl,
ITriggerOptions
    {

        private MetroFramework.MetroColorStyle style =
            MetroFramework.MetroColorStyle.Default;

        public MetroFramework.MetroColorStyle Style
        {
            set
            {
                style = value;
                rbDaily.Style = style;
                rbMonthy.Style = style;
                rbOnce.Style = style;
                rbWeekly.Style = style;
                weeklyTrigger.Style = style;
```

```
                monthyTrigger.Style = style;
            }
        }

        public TimeTrigger()
        {
            InitializeComponent();

            rbDaily.CheckedChanged += RbDaily_CheckedChanged;
            rbWeekly.CheckedChanged += RbWeekly_CheckedChanged;
            rbMonthy.CheckedChanged += RbMonthy_CheckedChanged;
        }

        private void RbMonthy_CheckedChanged(object sender,
EventArgs e)
        {
            if (rbMonthy.Checked)
                monthyTrigger.Visible = true;
            else
                monthyTrigger.Visible = false;
        }

        private void RbWeekly_CheckedChanged(object sender,
EventArgs e)
        {
            if (rbWeekly.Checked)
                weeklyTrigger.Visible = true;
            else
                weeklyTrigger.Visible = false;
        }

        private void RbDaily_CheckedChanged(object sender,
EventArgs e)
        {
            if (rbDaily.Checked)
                dailyTrigger.Visible = true;
```

```
            else
                dailyTrigger.Visible = false;
        }

        public TaskEditor.Trigger GetOptions()
        {
            TaskEditor.Trigger trigger = null;

            return trigger;
        }

        public bool Verify()
        {
            if (rbWeekly.Checked)
            {
                return weeklyTrigger.Verify();
            }
            else if (rbMonthy.Checked)
            {
                return monthyTrigger.Verify();
            }

            return true;
        }
        public void ShowTrigger(TaskEditor.TimeTrigger
timeTrigger)
        {
            datePicker.Value = timeTrigger.startTime;
            timePicker.Value = timeTrigger.startTime;

            switch (timeTrigger.timeType)
            {
                case
TaskEditor.ETimeTriggerType.TIME_TRIGGER_ONCE:

                    rbOnce.Select();
```

```
                    break;

                case
TaskEditor.ETimeTriggerType.TIME_TRIGGER_WEEK:

                    rbWeekly.Checked = true;

weeklyTrigger.ShowTrigger((TaskEditor.WeeklyTrigger)timeTrigger);

                    break;
                case TaskEditor.ETimeTriggerType.TIME_TRIGGER_DAY:

                    rbDaily.Checked = true;

dailyTrigger.ShowTrigger((TaskEditor.DailyTrigger)timeTrigger);

                    break;
                case
TaskEditor.ETimeTriggerType.TIME_TRIGGER_MONTH:

                    rbMonthy.Checked = true;

monthyTrigger.ShowTrigger((TaskEditor.MonthyTrigger)timeTrigger);

                    break;

                default:
                    break;
            }
        }

        public TaskEditor.Trigger GetTriggerOptions()
        {
            TaskEditor.TimeTrigger timeTrigger = null;
```

```
            if (rbOnce.Checked)
            {
                timeTrigger = new TaskEditor.OnceTrigger();
            }
            else if (rbWeekly.Checked)
            {
                timeTrigger = weeklyTrigger.GetTrigger();
            }
            else if (rbDaily.Checked)
            {
                timeTrigger = dailyTrigger.GetTrigger();
            }
            else if (rbMonthy.Checked)
            {
                timeTrigger = monthyTrigger.GetTrigger();
            }

            timeTrigger.startTime = new DateTime(
                datePicker.Value.Year,
                datePicker.Value.Month,
                datePicker.Value.Day,
                timePicker.Value.Hour,
                timePicker.Value.Minute,
                timePicker.Value.Second);

            return timeTrigger;
        }
    }
}
```

Приложение А.5

Файл «Task.cs»

```
namespace TaskEditor
{
```

```
public class Task
{
    // Флаг работы задачи
    public bool isEnable;

    // Путь задачи
    public string path;

    // Имя задачи
    public string name;

    // Описание задачи
    public string description;

    /* Параметры задачи */
    public Settings settings;

    /* Параметры безопасности задачи */
    public Security security;

    /* Время предыдущего исполнения задачи */
    public DateTime lastExecute;

    /* Время следующего исполнения задачи */
    public DateTime nextExecute;

    [System.ComponentModel.DisplayName("Имя")]
    public string Name
    {
        get { return name; }
        set { name = value; }
    }

    [System.ComponentModel.DisplayName("Описание")]
    public string Description
    {
```

```
            get { return description; }
            set { description = value; }

        }

        [System.ComponentModel.DisplayName("Последний запуск")]
        public string LastExecutedTime
        {
            get
            {
                return lastExecute == DateTime.MaxValue ?
                    "Задача еще не была запущена" :
lastExecute.ToString();
            }
        }

        [System.ComponentModel.DisplayName("Время следующего
запуска")]
        public string NextExecuteTime
        {
            get
            {
                return nextExecute == DateTime.MaxValue ?
                   "Время следующего запуска не определено" :
nextExecute.ToString();
            }

        }

        /* Триггеры задачи */
        public List<Trigger> triggerList;

        /* Действия задачи */
        public List<Action> actionList;

        public Task()
```

```
        {
            this.security = new Security();
            this.settings = new Settings();

            this.settings.DeleteDisabledTaskIn =
DateTime.MaxValue;
            this.settings.TerminateExecutedProcessIn =
DateTime.MaxValue;

            this.isEnable = true;

            this.lastExecute = DateTime.MaxValue;
            this.nextExecute = DateTime.MaxValue;

            this.triggerList = new List<Trigger>();
            this.actionList = new List<Action>();
        }

        public List<SRequstData> ConvertToMessage()
        {
            List<SRequstData> result = new List<SRequstData>();
            UnmanagedTaskInfo taskInfo = new UnmanagedTaskInfo()
            {
                isActive = this.isEnable,

                name = taskInfo.name = this.name,

                description = this.description,

                nextExecuteTime = this.nextExecute.ToFileTime(),
                prevExecuteTime = this.lastExecute.ToFileTime()
            };

            result.Add(new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_TASK,
```

```
                BufferOfData =
ServerWorker.ConvertStructToByteArray<UnmanagedTaskInfo>(taskInfo)
            });

            UnmanagedSecurity securityInfo = new
UnmanagedSecurity()
            {
                userName = this.security.userName,
                userPassword = this.security.userPassword,

            };

            result.Add(new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_SECURITY,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<UnmanagedSecurity>(securityI
nfo)
            });


            UnmanagedSettings settingsInfo = new
UnmanagedSettings()
            {
                deleteDisabledTaskIn =
this.settings.DeleteDisabledTaskIn.ToFileTime(),
                terminateExecutedProcessIn =
this.settings.TerminateExecutedProcessIn.ToFileTime()
            };

            result.Add(new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_SETTINGS,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<UnmanagedSettings>(settingsI
nfo)
```

```
            });

            foreach (Trigger trigger in triggerList)
            {
                result.Add(trigger.ConvertToMessage());
            }

            foreach (Action action in actionList)
            {
                result.Add(action.ConvertToMessage());

            }

            return result;
        }

        public void Add()
        {

            List<SRequstData> response = null;

            List<SRequstData> messages = this.ConvertToMessage();

            SRequestInfo requstInfo = new SRequestInfo()
            {
                dwMessageCount = (byte)(messages.Count + 1),
                RequstInfoType = EReqResInfoType.REQUEST_INFO_ADD
            };

            try
            {
                response =
Service.SendAcyncRequestToService(requstInfo, messages);
            }
            catch (Exception err)
            {
```

```
                throw new WarningException("Ошибка при добавлении
задачи", err);
            }

            foreach (SRequstData responseData in response)
            {
                switch (responseData.dataType)
                {
                    case EReqResDataType.REQUEST_DATA_TASK:

                        UnmanagedTaskInfo taskInfo =

(UnmanagedTaskInfo)ServerWorker.ConvertByteArrayToStruct<Unmanaged
TaskInfo>(responseData.BufferOfData);

                        this.lastExecute =
DateTime.FromFileTime(taskInfo.prevExecuteTime);
                        this.nextExecute =
DateTime.FromFileTime(taskInfo.nextExecuteTime);

                        break;
                }
            }
        }

        public void Delete()
        {
            SRequestInfo requstInfo = new SRequestInfo()
            {
                dwMessageCount = 2,
                RequstInfoType =
EReqResInfoType.REQUEST_INFO_DELETE
            };
            UnmanagedTaskInfo taskInfo = new UnmanagedTaskInfo();

            taskInfo.name = this.name;
```

```
            List<SRequstData> messages = new List<SRequstData>();

            messages.Add(
                new SRequstData()
                {
                    dataType = EReqResDataType.REQUEST_DATA_TASK,
                    BufferOfData =
ServerWorker.ConvertStructToByteArray<UnmanagedTaskInfo>(taskInfo)
                });

            try
            {
                Service.SendAcyncRequestToService(requstInfo,
messages);
            }
            catch (Exception err)
            {
                throw new WarningException("Ошибка при удалении
задачи", err);
            }
        }

        public static List<Task> GetAll()
        {

            List<SRequstData> response = null;

            List<Task> newTasks = new List<Task>();
            List<Trigger> triggers = new List<Trigger>();
            List<Action> actions = new List<Action>();

            Task newTask = null;

            SRequestInfo requstInfo = new SRequestInfo()
            {
```

```
                dwMessageCount = 1,
                RequstInfoType =
ЕReqResInfoType.REQUEST_INFO_GETTASKS,
                details = new SRequestInfo.Details()
                {

                }
            };

            try
            {
                response =
Service.SendAcyncRequestToService(requstInfo, null);
            }
            catch (Exception err)
            {
                throw new WarningException("Ошибка при получении
списка задач", err);
            }

            foreach (SRequstData responseData in response)
            {
                switch (responseData.dataType)
                {

                    case EReqResDataType.REQUEST_DATA_TASK:

                        UnmanagedTaskInfo taskInfo =

(UnmanagedTaskInfo)ServerWorker.ConvertByteArrayToStruct<Unmanaged
TaskInfo>(responseData.BufferOfData);

                        newTask = new Task();

                        newTask.isEnable = taskInfo.isActive;
```

```
                        newTask.path =
Path.GetDirectoryName(taskInfo.name);

                        newTask.path = newTask.path.Replace('\\',
'/');

                        newTask.name =
Path.GetFileName(taskInfo.name);
                        newTask.description =
taskInfo.description;

                        newTask.nextExecute =
DateTime.FromFileTime(taskInfo.nextExecuteTime);
                        newTask.lastExecute =
DateTime.FromFileTime(taskInfo.prevExecuteTime);

                        newTask.triggerList = triggers;
                        newTask.actionList = actions;

                        break;

                    case EReqResDataType.REQUEST_DATA_SECURITY:

                        UnmanagedSecurity securityInfo =

(UnmanagedSecurity)ServerWorker.ConvertByteArrayToStruct<Unmanaged
Security>(responseData.BufferOfData);

                        newTask.security.userName =
securityInfo.userName;
                        newTask.security.userPassword =
securityInfo.userPassword;

                        break;
```

```
                    case EReqResDataType.REQUEST_DATA_TRIGGER:

                        STrigger triggerInfo =

(STrigger)ServerWorker.ConvertByteArrayToStruct<STrigger>(response
Data.BufferOfData);

                        switch (triggerInfo.triggerType)
                        {
                            case ETriggerType.TRIGGER_TIME:
                                switch
(triggerInfo.triggerUnion.time.timeTriggerType)
                                {
                                    case
ETimeTriggerType.TIME_TRIGGER_ONCE:

                                        triggers.Add(new
OnceTrigger()
                                        {
                                            isEnable =
triggerInfo.isEnable,
                                            workTime =
DateTime.FromFileTime(triggerInfo.workTime),
                                            startTime =
DateTime.FromFileTime(triggerInfo.triggerUnion.time.startTime),
                                            isWorkTimeEnable =
triggerInfo.isWorkTimeEnable

                                        });
                                        break;

                                    case
ETimeTriggerType.TIME_TRIGGER_WEEK:
                                        triggers.Add(new
WeeklyTrigger()
                                        {
```

```
                                                isEnable =
triggerInfo.isEnable,
                                                daysOfWeek =
triggerInfo.triggerUnion.time.triggerData.weekly.week,
                                                frequency =
triggerInfo.triggerUnion.time.triggerData.weekly.frequency,
                                                workTime =
DateTime.FromFileTime(triggerInfo.workTime),
                                                startTime =
DateTime.FromFileTime(triggerInfo.triggerUnion.time.startTime),
                                                isWorkTimeEnable =
triggerInfo.isWorkTimeEnable

                                            });
                                            break;
                                        case
ETimeTriggerType.TIME_TRIGGER_DAY:
                                            triggers.Add(new
DailyTrigger()
                                            {
                                                isEnable =
triggerInfo.isEnable,
                                                frequency =
triggerInfo.triggerUnion.time.triggerData.daily.frequency,
                                                workTime =
DateTime.FromFileTime(triggerInfo.workTime),
                                                startTime =
DateTime.FromFileTime(triggerInfo.triggerUnion.time.startTime),
                                                isWorkTimeEnable =
triggerInfo.isWorkTimeEnable

                                            });
                                            break;
                                        case
ETimeTriggerType.TIME_TRIGGER_MONTH:
```

```
                                        switch
(triggerInfo.triggerUnion.time.triggerData.month.monthTriggerType)
                                        {

                                            case
ETimeTriggerType.TIME_TRIGGER_WEEK:
                                                triggers.Add(new
WeeklyMonthTrigger()
                                                {
                                                    orderInWeek =
triggerInfo.triggerUnion.time.triggerData.month.subType.weekOfMont
h.orderInWeek,
                                                    daysOfWeek =
triggerInfo.triggerUnion.time.triggerData.month.subType.weekOfMont
h.daysOfWeek,
                                                    last =
triggerInfo.triggerUnion.time.triggerData.month.subType.weekOfMont
h.last,
                                                    isEnable =
triggerInfo.isEnable,
                                                    months =
triggerInfo.triggerUnion.time.triggerData.month.month,
                                                    workTime =
DateTime.FromFileTime(triggerInfo.workTime),
                                                    startTime =
DateTime.FromFileTime(triggerInfo.triggerUnion.time.startTime),

isWorkTimeEnable = triggerInfo.isWorkTimeEnable

                                                });
                                                break;

                                            case
ETimeTriggerType.TIME_TRIGGER_DAY:
                                                triggers.Add(new
DailyMonthTrigger()
```

```
                                                {
                                                    Days =
triggerInfo.triggerUnion.time.triggerData.month.subType.daysOfMont
h.days,
                                                    last =
triggerInfo.triggerUnion.time.triggerData.month.subType.daysOfMont
h.last,
                                                    isEnable =
triggerInfo.isEnable,
                                                    months =
triggerInfo.triggerUnion.time.triggerData.month.month,
                                                    workTime =
DateTime.FromFileTime(triggerInfo.workTime),
                                                    startTime =
DateTime.FromFileTime(triggerInfo.triggerUnion.time.startTime),

isWorkTimeEnable = triggerInfo.isWorkTimeEnable

                                                });
                                                break;
                                            default:
                                                break;
                                        }

                                        break;
                                    case
ETimeTriggerType.TIME_TRIGGER_HOURS:
                                        break;
                                    case
ETimeTriggerType.TIME_TRIGGER_PEREODIC:
                                        break;
                                    default:
                                        break;
                                }

                                break;
```

```
                            case ETriggerType.TRIGGER_LOGON:
                                break;
                            case ETriggerType.TRIGGER_IDLE:
                                break;
                            case ETriggerType.TRIGGER_SYSTEMSTART:
                                break;
                            case ETriggerType.TRIGGER_LOGOUT:
                                break;
                            case ETriggerType.TRIGGER_ENABLE:
                                break;
                            default:
                                break;
                        }


                        break;
                    case EReqResDataType.REQUEST_DATA_ACTION:

                        SAction actionInfo =

(SAction)ServerWorker.ConvertByteArrayToStruct<SAction>(responseDa
ta.BufferOfData);

                        switch (actionInfo.actionType)
                        {
                            case
EActionType.ACTION_EXECUTE_PROGRAMM:

                                SActionExecuteProgramm
actionExecuteProgramm =

(SActionExecuteProgramm)ServerWorker.ConvertByteArrayToStruct<SAct
ionExecuteProgramm>
                                (actionInfo.BufferOfData);

                                actions.Add(
```

```
                                    new ProgrammExecuteAction()
                                    {
                                        isEnable =
actionInfo.isEnable,
                                        workTime =
DateTime.FromFileTime(actionInfo.workTime),
                                        isInteractiveExecute =
actionExecuteProgramm.isInteractiveExecute,
                                        programmPath =
actionExecuteProgramm.szProgrammPath,
                                        programmArgs =
actionExecuteProgramm.szArgs
                                    }
                                    );

                                    break;
                                case EActionType.ACTION_EMAIL_SEND:
                                    break;
                                case EActionType.ACTION_SHELL_MESSAGE:
                                    break;
                                default:
                                    break;
                            }


                        break;
                    case EReqResDataType.REQUEST_DATA_SETTINGS:

                        UnmanagedSettings settingsInfo =

(UnmanagedSettings)ServerWorker.ConvertByteArrayToStruct<Unmanaged
Settings>(responseData.BufferOfData);

                        newTask.settings.DeleteDisabledTaskIn =

DateTime.FromFileTime(settingsInfo.deleteDisabledTaskIn);
```

```
newTask.settings.TerminateExecutedProcessIn =

DateTime.FromFileTime(settingsInfo.terminateExecutedProcessIn);


                        break;

                    case EReqResDataType.REQUEST_DATA_INFO:

                        if (newTask != null)
                        {
                            newTasks.Add(newTask);
                            triggers = new List<Trigger>();
                            actions = new List<Action>();

                            newTask = null;

                        }
                        break;
                    default:
                        break;
                }
            }


            return newTasks;
        }

    }
}
```

Приложение А.6

Файл «Action.cs»

```
namespace TaskEditor
{
    public abstract class Action
    {
        /* Активность */
        public bool isEnable;

        /* Тип действия */
        public EActionType actionType;

        /* Время работы*/
        public DateTime workTime;

        public Action(EActionType type)
        {
            this.actionType = type;
        }

        public abstract SRequstData ConvertToMessage();

    }

    public class ProgrammExecuteAction : Action
    {
        /* Возможность интерактивного запуска*/
        public bool isInteractiveExecute;

        /* Возможность интерактивного запуска*/
        public string programmPath;

        /* Возможность интерактивного запуска*/
        public string programmArgs;
```

```
        public ProgrammExecuteAction()
            : base(EActionType.ACTION_EXECUTE_PROGRAMM)
        {

        }

        public override string ToString()
        {
            return String.Format("Запуск: {0}",
this.programmPath);
        }

        public override SRequstData ConvertToMessage()
        {
            SActionExecuteProgramm programmAction = new
SActionExecuteProgramm()
            {
                isInteractiveExecute = this.isInteractiveExecute,
                szProgrammPath = this.programmPath,
                szArgs = this.programmArgs
            };

            SAction action = new SAction()
            {
                actionType = this.actionType,
                isEnable = this.isEnable,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<SActionExecuteProgramm>(prog
rammAction)
            };

            SRequstData result = new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_ACTION,
```

```
                BufferOfData =
ServerWorker.ConvertStructToByteArray<SAction>(action)
            };

            return result;
        }
    }
}
```

Приложение А.7

Файл «Trigger.cs»

```
namespace TaskEditor
{
    public abstract class Trigger
    {
        /* Тип триггера */
        public ETriggerType triggerType;

        /* Активность */
        public bool isEnable;

        [System.ComponentModel.DisplayName("Статус")]
        public string WorkStatus
        {
            get { return isEnable ? "Включен" : "Выключен"; }
        }

        [System.ComponentModel.DisplayName("Тип")]
        public string TriggerType
        {
            get { return this.GetStringType(); }
        }

        [System.ComponentModel.DisplayName("Описание")]
```

```
        public string Description
        {
            get { return this.ToString(); }
        }

        public abstract string GetStringType();

        /* Время работы */
        public DateTime workTime;

        /* Активно ли время работы*/
        public bool isWorkTimeEnable;

        public Trigger(ETriggerType type)
        {
            this.triggerType = type;
        }
        public abstract SRequstData ConvertToMessage();

    }

    public abstract class TimeTrigger : Trigger
    {
        /* Тип временного триггера*/
        public ETimeTriggerType timeType;

        /* Время запуска триггера*/
        public DateTime startTime;

        public TimeTrigger(ETimeTriggerType type)
            : base(ETriggerType.TRIGGER_TIME)
        {
            timeType = type;
        }

        public override SRequstData ConvertToMessage()
```

```
        {
            STrigger trigger = new STrigger()
            {
                isEnable = this.isEnable,
                triggerType = this.triggerType,
                workTime = this.workTime.ToFileTime(),
                isWorkTimeEnable = this.isWorkTimeEnable
            };

            trigger.triggerUnion.time = new UnmanagedTimeTrigger()
            {
                startTime = this.startTime.ToFileTime(),
                timeTriggerType = this.timeType
            };

            SRequstData result = new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_TRIGGER,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<STrigger>(trigger)
            };

            return result;
        }

        public override string ToString()
        {
            return String.Format("Время запуска: {0}", startTime);
        }

        public override string GetStringType()
        {
            return "Временной ";
        }
    }
    public class OnceTrigger : TimeTrigger
```

```
    {
        public OnceTrigger()
            : base(ETimeTriggerType.TIME_TRIGGER_ONCE)
        {

        }
        public override SRequstData ConvertToMessage()
        {
            return base.ConvertToMessage();
        }

        public override string ToString()
        {
            return String.Format("Тип запуска: Однажды. Время
запуска: {0}", startTime);
        }

        public override string GetStringType()
        {
            return "Временной, одноразовый";
        }
    }

    public class DailyTrigger : TimeTrigger
    {

        /* Частота запуска*/
        public UInt32 frequency;

        public DailyTrigger()
            : base(ETimeTriggerType.TIME_TRIGGER_DAY)
        {

        }

        public override string GetStringType()
```

```
        {
            return "Временной, ежедневный";
        }

        public override SRequstData ConvertToMessage()
        {

            STrigger trigger = new STrigger()
            {
                isEnable = this.isEnable,
                triggerType = this.triggerType,
                workTime = this.workTime.ToFileTime(),
                isWorkTimeEnable = this.isWorkTimeEnable

            };

            trigger.triggerUnion.time = new UnmanagedTimeTrigger()
            {
                startTime = this.startTime.ToFileTime(),
                timeTriggerType = this.timeType
            };

            trigger.triggerUnion.time.triggerData.daily.frequency
= this.frequency;

            SRequstData result = new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_TRIGGER,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<STrigger>(trigger)
            };

            return result;
        }

        public override string ToString()
```

```
        {
            return String.Format("Тип запуска: Ежедневно. Время
запуска: {0}", startTime);
        }

    }

    public class WeeklyTrigger : TimeTrigger
    {

        /* Дни недели в которые будут осуществятся запуск */
        public SDaysOfWeek daysOfWeek;
        public UInt32 frequency;

        public WeeklyTrigger()
            : base(ETimeTriggerType.TIME_TRIGGER_WEEK)
        {
        }

        public override string GetStringType()
        {
            return "Временной, еженедельный";
        }

        public override SRequstData ConvertToMessage()
        {

            STrigger trigger = new STrigger()
            {
                isEnable = this.isEnable,
                triggerType = this.triggerType,
                workTime = this.workTime.ToFileTime(),
                isWorkTimeEnable = this.isWorkTimeEnable

            };
```

```
            trigger.triggerUnion.time = new UnmanagedTimeTrigger()
            {
                startTime = this.startTime.ToFileTime(),
                timeTriggerType = this.timeType
            };

            trigger.triggerUnion.time.triggerData.weekly.week =
this.daysOfWeek;
            trigger.triggerUnion.time.triggerData.weekly.frequency
= this.frequency;



            SRequstData result = new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_TRIGGER,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<STrigger>(trigger)
            };

            return result;
        }

        public override string ToString()
        {
            return String.Format("Тип запуска: Еженедельно. Время
запуска: {0}", startTime);
        }
    }

    public abstract class MonthyTrigger : TimeTrigger
    {

        /* Месяцы для запуска задачи*/
        public SMonths months;

        /* Тип месячного триггера*/
```

```
        public ETimeTriggerType monthTriggerType;

        public MonthyTrigger(ETimeTriggerType MonthTriggerType)
            : base(ETimeTriggerType.TIME_TRIGGER_MONTH)
        {
            this.monthTriggerType = MonthTriggerType;
        }

        public override SRequstData ConvertToMessage()
        {

            STrigger trigger = new STrigger()
            {
                isEnable = this.isEnable,
                triggerType = this.triggerType,
                workTime = this.workTime.ToFileTime(),
                isWorkTimeEnable = this.isWorkTimeEnable

            };

            trigger.triggerUnion.time = new UnmanagedTimeTrigger()
            {
                startTime = this.startTime.ToFileTime(),
                timeTriggerType = this.timeType
            };


trigger.triggerUnion.time.triggerData.month.monthTriggerType =
this.monthTriggerType;
            trigger.triggerUnion.time.triggerData.month.month =
this.months;

            SRequstData result = new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_TRIGGER,
```

```
                BufferOfData =
ServerWorker.ConvertStructToByteArray<STrigger>(trigger)
            };

            return result;
        }
        public override string ToString()
        {
            return String.Format("Тип запуска: Ежемесячно. Время
запуска: {0}", startTime);
        }

        public override string GetStringType()
        {
            return "Временной, ежемесячный";
        }
    }

    public class WeeklyMonthTrigger : MonthyTrigger
    {

        public bool[] orderInWeek;

        public SDaysOfWeek daysOfWeek;

        public bool last;

        public WeeklyMonthTrigger()
            : base(ETimeTriggerType.TIME_TRIGGER_WEEK)
        {
            orderInWeek = new bool[31];
        }

        public override SRequstData ConvertToMessage()
        {
```

```
            STrigger trigger = new STrigger()
            {
                isEnable = this.isEnable,
                triggerType = this.triggerType,
                workTime = this.workTime.ToFileTime(),
                isWorkTimeEnable = this.isWorkTimeEnable
            };

            trigger.triggerUnion.time = new UnmanagedTimeTrigger()
            {
                startTime = this.startTime.ToFileTime(),
                timeTriggerType = this.timeType
            };


trigger.triggerUnion.time.triggerData.month.monthTriggerType =
this.monthTriggerType;
            trigger.triggerUnion.time.triggerData.month.month =
this.months;

trigger.triggerUnion.time.triggerData.month.subType.weekOfMonth.or
derInWeek = this.orderInWeek;

trigger.triggerUnion.time.triggerData.month.subType.weekOfMonth.da
ysOfWeek = this.daysOfWeek;

trigger.triggerUnion.time.triggerData.month.subType.weekOfMonth.la
st = this.last;

            SRequstData result = new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_TRIGGER,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<STrigger>(trigger)
            };
```

```
            return result;
        }

        public override string ToString()
        {
            return String.Format("Тип запуска: Ежемесячно, по дням
недели. Время запуска: {0}", startTime);
        }

        public override string GetStringType()
        {
            return "Временной, ежемесячный по неделям";
        }
    }

    public class DailyMonthTrigger : MonthyTrigger
    {

        public bool[] Days;

        public bool last;

        public DailyMonthTrigger()
            : base(ETimeTriggerType.TIME_TRIGGER_DAY)
        {
            Days = new bool[31];
        }

        public override SRequstData ConvertToMessage()
        {

            STrigger trigger = new STrigger()
            {
                isEnable = this.isEnable,
                triggerType = this.triggerType,
                workTime = this.workTime.ToFileTime(),
```

```
                isWorkTimeEnable = this.isWorkTimeEnable
            };

            trigger.triggerUnion.time = new UnmanagedTimeTrigger()
            {
                startTime = this.startTime.ToFileTime(),
                timeTriggerType = this.timeType
            };


trigger.triggerUnion.time.triggerData.month.monthTriggerType =
this.monthTriggerType;
            trigger.triggerUnion.time.triggerData.month.month =
this.months;

trigger.triggerUnion.time.triggerData.month.subType.daysOfMonth.da
ys = this.Days;

trigger.triggerUnion.time.triggerData.month.subType.daysOfMonth.la
st = this.last;


            SRequstData result = new SRequstData()
            {
                dataType = EReqResDataType.REQUEST_DATA_TRIGGER,
                BufferOfData =
ServerWorker.ConvertStructToByteArray<STrigger>(trigger)
            };

            return result;
        }
        public override string ToString()
        {
            return String.Format("Тип запуска: Ежемесячно, по дням
месяца. Время запуска: {0}", startTime);
        }
```

```
        public override string GetStringType()
        {
            return "Временной, ежемесячный по дням";
        }
    }
}
```

Приложение А.8

Файл «ServerWorker.cs»

```
namespace TaskEditor
{
    class ServerWorker
    {
        public static byte[] ConvertStructToByteArray<T>(T
sStruct) where T : struct
        {

            int structSize = Marshal.SizeOf(sStruct);
            byte[] arr = new byte[structSize];

            IntPtr ptr = Marshal.AllocHGlobal(structSize);
            Marshal.StructureToPtr(sStruct, ptr, true);
            Marshal.Copy(ptr, arr, 0, structSize);
            Marshal.FreeHGlobal(ptr);

            return arr;
        }

        public static T ConvertByteArrayToStruct<T>(byte[] bytes)
where T : struct
        {
            GCHandle handle = GCHandle.Alloc(bytes,
GCHandleType.Pinned);
```

```
            T stuff =
(T)Marshal.PtrToStructure(handle.AddrOfPinnedObject(), typeof(T));

            handle.Free();

            return stuff;
        }

    }

    public class Service
    {

        public static string hostIp = "127.0.0.1";

        public ServiceController controller;

        public Service(string serviceName)
        {
            controller = new ServiceController(serviceName);
        }

        public void Start()
        {
            controller.Refresh();

            if (controller.Status !=
ServiceControllerStatus.Running &&
                controller.Status !=
ServiceControllerStatus.StartPending)
                controller.Start();
        }

        public void Stop()
        {
            controller.Refresh();
```

```
            if (controller.Status !=
ServiceControllerStatus.Stopped &&
                controller.Status !=
ServiceControllerStatus.StopPending)
                controller.Stop();
        }

        public ServiceControllerStatus GetStatus()
        {
            controller.Refresh();
            return controller.Status;
        }

        public override string ToString()
        {
            controller.Refresh();
            string statusString = null;

            switch (controller.Status)
            {
                case ServiceControllerStatus.ContinuePending:
                    statusString = "Ожидание продолжения.";
                    break;
                case ServiceControllerStatus.PausePending:
                    statusString = "Ожидание паузы.";
                    break;
                case ServiceControllerStatus.Paused:
                    statusString = "Пауза.";
                    break;
                case ServiceControllerStatus.Running:
                    statusString = "Работает.";
                    break;
                case ServiceControllerStatus.StartPending:
                    statusString = "Ожидание начала работы.";
                    break;
                case ServiceControllerStatus.StopPending:
```

```
                    statusString = "Ожидание остановки.";
                    break;
                case ServiceControllerStatus.Stopped:
                    statusString = "Остановлена.";
                    break;
            }

            return statusString;
        }

        /// <summary>
        /// Асинхронная отправка запроса службе
        /// </summary>
        /// <param name="requstInfo">Загаловок запроса</param>
        /// <param name="messages">Сообщения (данные
запроса)</param>
        /// <returns>Список данных полученных от службы</returns>
        public static List<SRequstData> SendAcyncRequestToService(
            SRequestInfo requstInfo,
            List<SRequstData> messages)
        {
            // Прогресс задачи
            TaskViewer.ProgressDialog progress = new
TaskViewer.ProgressDialog();

            // Ответ из службы
            List<SRequstData> response = new List<SRequstData>();

            // Исключение в процессе запроса
            Exception workException = null;

            Thread workThread = new Thread(
                new ThreadStart(() =>
                {
                    try
                    {
```

```
                        response =
SendRequestToService(requstInfo, messages);
                    }
                    catch (Exception err)
                    {
                        workException = err;
                    }

                    progress.BeginInvoke(new System.Action(() => {
progress.Close(); }));

                }));

            workThread.Start();

            progress.ShowDialog();

            if (workException != null)
            {
                throw workException;
            }

            return response;
        }

        /// <summary>
        /// Отправка запроса в службу
        /// </summary>
        /// <param name="requstInfo">Загаловок запроса</param>
        /// <param name="messages">Сообщения (данные
запроса)</param>
        /// <returns>Список данных полученных от службы</returns>
        public static List<SRequstData> SendRequestToService(
            SRequestInfo requstInfo,
            List<SRequstData> messages)
        {
```

```
            SRequestInfo responseDataInfo;
            List<SRequstData> response = new List<SRequstData>();

            byte[] dataSend;
            byte[] dataRecive = new byte[8192];

            // Открытие именованного канала для отсылки запроса
            using (NamedPipeClientStream pipeClient = new
NamedPipeClientStream(".", @"TASKPIPE"))
            {
                dataSend =
ServerWorker.ConvertStructToByteArray<SRequestInfo>(requstInfo);

                try
                {
                    // Соединение по именованном каналу
                    pipeClient.Connect(11000);

                }
                catch (System.TimeoutException)
                {
                    throw new System.TimeoutException("Истекло
время ожидания подключения к службе." +
                        Environment.NewLine +
                        "Попробуйте повторить попытку позже.");
                }

                // Запись в именованный канал
                if (pipeClient.CanWrite)
                    pipeClient.Write(dataSend, 0,
dataSend.Length);

                // Ожидание что будет считано всё сообщение
                pipeClient.WaitForPipeDrain();

                if (messages != null)
```

```
                    for (int i = 0; i < messages.Count; i++)
                    {
                        dataSend =
ServerWorker.ConvertStructToByteArray<SRequstData>(messages[i]);

                        // Запись в именованный канал
                        if (pipeClient.CanWrite)
                            pipeClient.Write(dataSend, 0,
dataSend.Length);

                        // Ожидание что будет считано всё
сообщение
                        pipeClient.WaitForPipeDrain();

                    }

                int readLegth = 0;

                /* Получение ответа */
                if (pipeClient.CanRead)
                    readLegth = pipeClient.Read(dataRecive, 0,
dataRecive.Length);

                if (readLegth == 0)
                    return response;

                SRequstData repData =
(SRequstData)ServerWorker.ConvertByteArrayToStruct<SRequstData>(da
taRecive);
                responseDataInfo =
(SRequestInfo)ServerWorker.ConvertByteArrayToStruct<SRequestInfo>(
repData.BufferOfData);

                int unMessageCount =
responseDataInfo.dwMessageCount;
```

```
                for (int i = 0; i < unMessageCount; i++)
                {
                    if (pipeClient.CanRead)
                        pipeClient.Read(dataRecive, 0,
dataRecive.Length);

                    response.Add(

ServerWorker.ConvertByteArrayToStruct<SRequstData>(dataRecive));

                    repData =
(SRequstData)ServerWorker.ConvertByteArrayToStruct<SRequstData>(da
taRecive);

                    switch (repData.dataType)
                    {

                        case EReqResDataType.REQUEST_DATA_INFO:

                            SRequestInfo info =

(SRequestInfo)ServerWorker.ConvertByteArrayToStruct<SRequestInfo>(
repData.BufferOfData);

                            if (info.RequstInfoType ==
EReqResInfoType.RESPONSE_INFO_ERROR)
                            {
                                throw new Exception(
                                    String.Format("Ошибка в
результате выполнения запроса, код: {0}",

info.details.errorInfo.errorCode));
                            }

                            unMessageCount += info.dwMessageCount;
```

```
                            break;
                        default:
                            break;
                    }
                }
                return response;
            }
        }
    }
}
```

Приложение А.9

Файл « Program.cs»

```
namespace TaskEditor
{
    static class Program
    {
        public static Service taskPlannerSrv;

        [STAThread]
        static void Main()
        {
            taskPlannerSrv = new Service("TaskPlanner");

            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new MainForm());
        }
    }
}
```

Приложение В

Листинги программы сервера

Приложение В.1

Файл «Service.c»

```
/**
* @file Service.c
* Планировщик задач
*/
#include "stdafx.h"

extern SERVICE_STATUS             g_ServiceStatus = { 0 };
extern SERVICE_STATUS_HANDLE  g_StatusHandle = NULL;
extern HANDLE                         g_ServiceStopEvent =
INVALID_HANDLE_VALUE;             // Событие остановки службы


//=================================================================
=============
VOID
WINAPI
ServiceMain(DWORD argc, LPTSTR* argv){

     DWORD Status = E_FAIL;
     DWORD dwError = ERROR_SUCCESS;


     g_StatusHandle = RegisterServiceCtrlHandlerEx(SERVICE_NAME,
(LPHANDLER_FUNCTION_EX)ServiceCtrlHandler, NULL);

     if (g_StatusHandle == NULL)
     {
          return;
     }
```

```
     __try
     {
          memset(&g_ServiceStatus, 0, sizeof(g_ServiceStatus));
          g_ServiceStatus.dwServiceType =
SERVICE_INTERACTIVE_PROCESS;
          g_ServiceStatus.dwServiceType =
SERVICE_WIN32_OWN_PROCESS;
          g_ServiceStatus.dwControlsAccepted = 0;
          g_ServiceStatus.dwCurrentState = SERVICE_START_PENDING;
          g_ServiceStatus.dwWin32ExitCode = 0;
          g_ServiceStatus.dwServiceSpecificExitCode = 0;
          g_ServiceStatus.dwCheckPoint = 0;

          if (SetServiceStatus(g_StatusHandle, &g_ServiceStatus) ==
FALSE){
          }

          g_ServiceStopEvent = CreateEvent(NULL, TRUE, FALSE,
NULL);

          if (g_ServiceStopEvent == NULL){
               g_ServiceStatus.dwControlsAccepted = 0;
               g_ServiceStatus.dwCurrentState = SERVICE_STOPPED;
               g_ServiceStatus.dwWin32ExitCode = GetLastError();
               g_ServiceStatus.dwCheckPoint = 1;
               if (SetServiceStatus(g_StatusHandle,
&g_ServiceStatus) == FALSE){
               }
          }
          g_ServiceStatus.dwControlsAccepted = SERVICE_ACCEPT_STOP;
          g_ServiceStatus.dwCurrentState = SERVICE_RUNNING;
          g_ServiceStatus.dwWin32ExitCode = 0;
          g_ServiceStatus.dwCheckPoint = 2;
```

```
        if (SetServiceStatus(g_StatusHandle, &g_ServiceStatus) ==
FALSE)
        {

        }

        /* Для отладки*/
        Sleep(10000);

        /* Создание списка таймеров*/
        if ((dwError = TimersContextCreate()) != ERROR_SUCCESS){
             return;
        }

        /* Создание списка задач*/
        if ((dwError = TaskHashTableCreate(&g_pTaskHashTable)) !=
ERROR_SUCCESS){
             return;
        }

        /* Получение списка задач из реестра */
        if ((dwError = GetTasksFromRegistry(g_pTaskHashTable)) !=
ERROR_SUCCESS){
             return;
        }

        /* Установка триггеров */
        if ((dwError = TaskTriggersInstallAll()) !=
ERROR_SUCCESS) {
             return;
        }


        /* Создание нити ожидающая срабатывания таймеров задач */
        HANDLE hWaitTimersThread = CreateThread(NULL,
0,(LPTHREAD_START_ROUTINE) WaitForTimersThread, NULL, 0, NULL);
```

```
            /* Создание нити для обработки запросов пользователя */
            HANDLE hClientRequestHandlerThread = CreateThread(NULL,
0, (LPTHREAD_START_ROUTINE)ClientRequestReciver, NULL, 0, NULL);

            // Ожидание завершения всех нитей
            WaitForSingleObject(hWaitTimersThread, INFINITE);
            WaitForSingleObject(hClientRequestHandlerThread,
INFINITE);

      }
      __finally{

            // Освобождение ресурсов
            g_ServiceStatus.dwControlsAccepted = 0;
            g_ServiceStatus.dwCurrentState = SERVICE_STOPPED;
            g_ServiceStatus.dwWin32ExitCode = 0;
            g_ServiceStatus.dwCheckPoint = 3;


            if (SetServiceStatus(g_StatusHandle, &g_ServiceStatus) ==
FALSE){

            }
      }
}

//=============================================================
=============
VOID
WINAPI
ServiceCtrlHandler(DWORD CtrlCode){
      switch (CtrlCode)
      {
      case SERVICE_CONTROL_STOP:
```

```
          if (g_ServiceStatus.dwCurrentState != SERVICE_RUNNING)
               break;

          g_ServiceStatus.dwControlsAccepted = 0;
          g_ServiceStatus.dwCurrentState = SERVICE_STOP_PENDING;
          g_ServiceStatus.dwWin32ExitCode = 0;
          g_ServiceStatus.dwCheckPoint = 4;

          SetEvent(g_ServiceStopEvent);

          break;
     default:
          break;
     }
}
```

Приложение В.2

Файл « Task.c»

```
/**
* @file Task.c
* Планировщик задач
*/

#include "stdafx.h"

extern PSTaskHashTable g_pTaskHashTable = NULL;

BOOL
CompareTasks(
     _In_ PSTask pTask1,
     _In_ PSTask pTask2
)
{
```

```
    if (_wcsicmp(pTask1->szTaskName, pTask2->szTaskName) == 0)
        return TRUE;
    else
        return FALSE;
}

DWORD
TaskCreate(
    _Inout_ PSTask* pTask
)
{
    *pTask = malloc(sizeof(STask));

    if (*pTask == NULL)
        return ERROR_NOT_ENOUGH_MEMORY;

    memset(*pTask, 0, sizeof(STask));

    return ERROR_SUCCESS;
}

//=============================================================
===============
DWORD
TaskHashTableCreate(
    _Out_ PSTaskHashTable* ppTaskHashTable
)
{
    DWORD dwError = ERROR_SUCCESS;

    if ((dwError = LinkedHashTableCreate(CompareTasks,
ppTaskHashTable)) != ERROR_SUCCESS)
    {
        return dwError;
    }
```

```
     return ERROR_SUCCESS;
}

//================================================================
===============
DWORD
TaskHashTableAddTask(
     _Inout_ PSTaskHashTable pTaskHashTable,
     _In_ PSTask pTask
)
{
     DWORD dwError = ERROR_SUCCESS;
     PSGenericElement pTaskElem = NULL;

     if (pTaskHashTable == NULL || pTask == NULL)
          return ERROR_INVALID_PARAMETER;

     if ((dwError = GenericElementCreate(&pTaskElem)) !=
ERROR_SUCCESS)
     {
          return dwError;
     }
     __try
     {
          pTaskElem->pData = pTask;
          pTaskElem->eElementType = TASK_TYPE;

          if ((dwError = LinkedHashTableAddElement(
               pTaskHashTable,
               pTaskElem,
               pTask->szTaskName,
               (wcslen(pTask->szTaskName) + 1) * sizeof(WCHAR)))
               != ERROR_SUCCESS)
          {
               return dwError;
          }
```

```
	}
	__finally
	{
		if (AbnormalTermination())
		{
			GenericElementFree(pTaskElem);
		}
	}

	return ERROR_SUCCESS;
}

//================================================================
================
DWORD
TaskHashTableFindTask(
	_In_ PSTaskHashTable pTaskHashTable,
	_In_ WCHAR* szName,
	_Out_ PSTask* ppTask
)
{
	DWORD dwError = ERROR_SUCCESS;
	PSTask pSearchTask = NULL;
	PSGenericElement pFoundElem = NULL;
	PSGenericElement pSearchElem = NULL;

	*ppTask = NULL;

	__try
	{

		if ((dwError = TaskCreate(&pSearchTask)) !=
ERROR_SUCCESS)
		{
			return dwError;
		}
```

```
          memcpy(pSearchTask->szTaskName, szName,
USTRING_SIZE(szName));

          if ((dwError = GenericElementCreate(&pSearchElem)) !=
ERROR_SUCCESS)
          {
               return dwError;
          }

          pSearchElem->eElementType = TASK_TYPE;
          pSearchElem->pData = pSearchTask;


          if ((dwError = LinkedTableFindElement(
               pTaskHashTable,
               szName,
               USTRING_SIZE(szName),
               pSearchElem,
               &pFoundElem)) != ERROR_SUCCESS)
          {
               return dwError;
          }

     }
     __finally
     {
          if (AbnormalTermination())
          {
               TaskPlannerFree(pSearchTask);
               GenericElementFree(pSearchElem);
          }
     }

     *ppTask = pFoundElem->pData;
```

```
     return ERROR_SUCCESS;
}

//================================================================
===============
DWORD
TaskHashTableDeleteTask(
     _In_ PSTaskHashTable pTaskHashTable,
     _In_ PSTask pTask
)
{
     DWORD dwError = ERROR_SUCCESS;

     PSGenericElement pDeleteElem = NULL;

     __try
     {
          if ((dwError = GenericElementCreate(&pDeleteElem)) !=
ERROR_SUCCESS)
          {
               return dwError;
          }

          pDeleteElem->eElementType = TASK_TYPE;
          pDeleteElem->pData = pTask;

          if ((dwError = LinkedTableDeleteElememt(
               pTaskHashTable,
               pTask->szTaskName,
               USTRING_SIZE(pTask->szTaskName),
               pDeleteElem)) != ERROR_SUCCESS)
          {
               return dwError;
          }
     }
     __finally
```

```
	{
		if (AbnormalTermination())
		{
			GenericElementFree(pDeleteElem);
		}
	}
	return ERROR_SUCCESS;
}

//================================================================
================
DWORD TaskAdd(
	_In_ PSTask pTask
)
{
	DWORD dwError = ERROR_SUCCESS;

	/* Добавление задачи в общий список */
	if ((dwError = TaskHashTableAddTask(g_pTaskHashTable, pTask))
!= ERROR_SUCCESS)
	{
		return dwError;
	}

	/* Добавление задачи в общий список */
	if ((dwError = TaskTriggersInstall(pTask)) != ERROR_SUCCESS)
	{
		return dwError;
	}

	/* Сохранение задачи в реестр */
	if ((dwError = SaveTaskToRegistry(pTask)) != ERROR_SUCCESS)
	{
		return dwError;
	}
```

```
	return dwError;
}

//================================================================
===============
DWORD TaskDelete(
	_In_ PSTask pTask
)
{
	PSTask pFoundTask = NULL;
	DWORD dwError = ERROR_SUCCESS;


	/* Поиск удаляемой задачи */
	if ((dwError = TaskHashTableFindTask(g_pTaskHashTable, pTask-
>szTaskName, &pFoundTask)) != ERROR_SUCCESS)
	{
		return dwError;
	}

	/* Проверка что задача найдена */
	if (pFoundTask == NULL) {
		return dwError;
	}

	/* Добавление задачи в общий список */
	if ((dwError = TaskTriggersDeinstall(pFoundTask)) !=
ERROR_SUCCESS)
	{
		return dwError;
	}

	/* Удаление задачи из реестра */
	if ((dwError = DeleteTaskFromRegistry(pFoundTask)) !=
ERROR_SUCCESS)
	{
```

```
          return dwError;
     }

     /* Удаление задачи */
     if ((dwError = TaskHashTableDeleteTask(g_pTaskHashTable,
pFoundTask)) != ERROR_SUCCESS)
     {
          return dwError;
     }

     TaskPlannerFree(pFoundTask->stTringgerList.pTriggers);
     TaskPlannerFree(pFoundTask->ActionList.pActions);
     TaskPlannerFree(pFoundTask);

     return dwError;
}

//=============================================================
================
DWORD
TaskEdit(
     _In_ PSTask pTask
)
{
     DWORD dwError = ERROR_SUCCESS;

     if ((dwError = TaskDelete(pTask)) != ERROR_SUCCESS)
     {
          return dwError;
     }

     if ((dwError = TaskHashTableAddTask(g_pTaskHashTable, pTask))
!= ERROR_SUCCESS)
     {
          return dwError;
     }
```

```
	return ERROR_SUCCESS;
}

//=================================================================
===============
DWORD
TaskTriggersInstall(
	_In_ PSTask pTask
)
{
	DWORD dwError = ERROR_SUCCESS;
	STriggerList pTriggerList = pTask->stTringgerList;

	for (DWORD i = 0; i < pTriggerList.dwCount; i++)
	{
		PSTrigger pTrigger = pTriggerList.pTriggers[i];

		if (pTrigger->isEnable)
			if ((dwError = TriggerInstall(pTask, pTrigger)) !=
ERROR_SUCCESS)
			{
				return dwError;
			}
	}

	return ERROR_SUCCESS;
}

//=================================================================
===============
DWORD
TaskTriggersInstallAll()
{
	DWORD dwError = ERROR_SUCCESS;
	PSLinkedHashTableEntry pTaskEntry = NULL;
```

```
     LinkedHashTableGetFirstEntry(g_pTaskHashTable, &pTaskEntry);

     while (pTaskEntry != NULL)
     {
          if ((dwError = TaskTriggersInstall(pTaskEntry-
>pGenericElem->pData))
               != ERROR_SUCCESS) {
               return dwError;
          }

          LinkedHashTableGetNextEntry(g_pTaskHashTable, pTaskEntry,
&pTaskEntry);
     }

     return ERROR_SUCCESS;
}

//================================================================
================
DWORD
TriggerInstall(
     _In_ PSTask pTask,
     _In_ PSTrigger pTrigger
) {
     PSTimerDetails pDetails = NULL;

     LARGE_INTEGER timerTime = { 0 };
     SYSTEMTIME currentSystemTime = { 0 };

     GetSystemTime(&currentSystemTime);

     // Проверка что время работы триггера еще не вышло
     if (pTrigger->isWorkTimeEnable &&
          (CompareSystemAndFileTime(
               currentSystemTime, pTrigger->WorkTime) == -1))
```

```
    {
        return ERROR_SUCCESS;
    }

    /* Обработка триггера в соответствии с его типом */
    switch (pTrigger->TriggerType)
    {

    case TRIGGER_TIME:
    {
        HANDLE hTimer = INVALID_HANDLE_VALUE;

        STriggerTime timeTrrigger = pTrigger->Trigger.Time;

        DWORD dwError = ERROR_SUCCESS;
        FILETIME resultTime = { 0 };
        LARGE_INTEGER timerTime = { 0 };

        /* Обработка временного триггера в соответствии с его
типом */
        switch (timeTrrigger.TimeTriggerType)
        {

        case TIME_TRIGGER_ONCE:

            GetSystemTime(&currentSystemTime);

            if ((CompareSystemAndFileTime(currentSystemTime,
timeTrrigger.StartTime)) == 1) {

                hTimer = CreateWaitableTimer(NULL, FALSE,
NULL);

                memcpy(&timerTime, &timeTrrigger.StartTime,
sizeof(FILETIME));
```

```
                    SetWaitableTimer(hTimer, &timerTime, 0, NULL,
NULL, FALSE);

                    if ((dwError = TimerDetailsCreate(&pDetails))
!= ERROR_SUCCESS)
                    {
                         return dwError;
                    }

                    pDetails->pTask = pTask;
                    pDetails->pTrigger = pTrigger;

                    TimerContextAddTimer(pDetails, hTimer);

                    pTask->nextExecuteTime =
timeTrrigger.StartTime;
               }

               break;

          case TIME_TRIGGER_MONTH:

               switch (timeTrrigger.Month.MonthTriggerType)
               {

               case TIME_TRIGGER_DAY:

                    if ((dwError =
GetDailyMonthTriggerTime(&timeTrrigger, &resultTime)) !=
ERROR_SUCCESS)
                    {
                         return dwError;
                    }

                    hTimer = CreateWaitableTimer(NULL, FALSE,
NULL);
```

```
                    memcpy(&timerTime, &resultTime,
sizeof(FILETIME));

                    SetWaitableTimer(hTimer, &resultTime, 0, NULL,
NULL, FALSE);

                    if ((dwError = TimerDetailsCreate(&pDetails))
!= ERROR_SUCCESS)
                    {
                         return dwError;
                    }

                    pDetails->pTask = pTask;
                    pDetails->pTrigger = pTrigger;

                    TimerContextAddTimer(pDetails, hTimer);

                    pTask->nextExecuteTime = resultTime;

                    break;

               case TIME_TRIGGER_WEEK:

                    if ((dwError =
GetWeeklyMonthTrigger(&timeTrrigger, &resultTime)) !=
ERROR_SUCCESS)
                    {
                         return dwError;
                    }

                    hTimer = CreateWaitableTimer(NULL, FALSE,
NULL);
                    memcpy(&timerTime, &resultTime,
sizeof(FILETIME));
```

```
                    SetWaitableTimer(hTimer, &resultTime, 0, NULL,
NULL, FALSE);

                    if ((dwError = TimerDetailsCreate(&pDetails))
!= ERROR_SUCCESS)
                    {
                         return ERROR_NOT_ENOUGH_MEMORY;
                    }

                    pDetails->pTask = pTask;
                    pDetails->pTrigger = pTrigger;

                    TimerContextAddTimer(pDetails, hTimer);

                    pTask->nextExecuteTime = resultTime;

                    break;

               default:
                    break;
               }

               break;

          case TIME_TRIGGER_DAY:

               if ((dwError = GetDailyTriggerTime(&timeTrrigger,
&resultTime)) != ERROR_SUCCESS)
               {
                    return dwError;
               }

               hTimer = CreateWaitableTimer(NULL, FALSE, NULL);
               memcpy(&timerTime, &resultTime, sizeof(FILETIME));
```

```
                SetWaitableTimer(hTimer, &resultTime, 0, NULL, NULL,
FALSE);
                if ((dwError = TimerDetailsCreate(&pDetails)) !=
ERROR_SUCCESS)
                {
                     return dwError;
                }

                pDetails->pTask = pTask;
                pDetails->pTrigger = pTrigger;



                TimerContextAddTimer(pDetails, hTimer);

                pTask->nextExecuteTime = resultTime;

                break;

           case TIME_TRIGGER_WEEK:

                if ((dwError = GetWeeklyTriggerTime(&timeTrrigger,
&resultTime)) != ERROR_SUCCESS)
                {
                     return dwError;
                }

                hTimer = CreateWaitableTimer(NULL, FALSE, NULL);
                memcpy(&timerTime, &resultTime, sizeof(FILETIME));

                SetWaitableTimer(hTimer, &resultTime, 0, NULL, NULL,
FALSE);

                if ((dwError = TimerDetailsCreate(&pDetails)) !=
ERROR_SUCCESS)
                {
                     return dwError;
```

```
                }

                pDetails->pTask = pTask;
                pDetails->pTrigger = pTrigger;

                TimerContextAddTimer(pDetails, hTimer);

                pTask->nextExecuteTime = resultTime;

                break;

           default:
                break;
           }

           break;
     }
     default:
           break;
     }

     return ERROR_SUCCESS;
}

//================================================================
================
DWORD
TaskTriggersDeinstall(
     _In_ PSTask pTask
)
{
     PSTrigger pTrigger = NULL;
     DWORD dwError = ERROR_SUCCESS;

     for (DWORD i = 0; i < pTask->stTringgerList.dwCount; i++)
     {
```

```
          pTrigger = pTask->stTringgerList.pTriggers[i];

          switch (pTrigger->TriggerType)
          {

          case TRIGGER_TIME:
          {
               DWORD dwTimerIndex = 0;

               while ((TimerContextGetTimerIndexByTask(pTask,
&dwTimerIndex)) == ERROR_SUCCESS)
               {
                    if ((dwError =
TimerContextDeleteTimer(dwTimerIndex)) != ERROR_SUCCESS)
                    {
                         return dwError;
                    }
               }

               break;
          }
          default:
               break;
          }
     }

     return ERROR_SUCCESS;
}

//================================================================
===============
DWORD TaskConvertToMessages(
     _In_ PSTask pTask,
     _Out_ PSMessageArray pMessageList
)
{
```

```
    DWORD dwError = ERROR_SUCCESS;

    PSMessage pMessage = NULL;
    SReqResData responseData;

    /* Преобразование общих параметров задачи в сообщение*/
    responseData.dataType = REQRES_DATA_TASK;
    responseData.Data.GeneralTask = *pTask;


    if ((dwError = MessageCreate(&responseData,
         sizeof(responseData),
         &pMessage)) != ERROR_SUCCESS)
    {
         return dwError;
    }

    if ((dwError = MessageArrayAddMessage(pMessageList, pMessage))
!= ERROR_SUCCESS)
    {
         return dwError;
    }

    /* Преобразование настроек задачи в сообщение*/

    responseData.dataType = REQRES_DATA_SETTINGS;
    responseData.Data.Setting = pTask->stSettings;

    if ((dwError = MessageCreate(&responseData,
         sizeof(responseData),
         &pMessage)) != ERROR_SUCCESS)
    {
         return dwError;
    }
```

```
     if ((dwError = MessageArrayAddMessage(pMessageList, pMessage))
!= ERROR_SUCCESS)
     {
          return dwError;
     }


     /* Преобразование настроек безопасности в сообщение*/
     responseData.dataType = REQRES_DATA_SECURITY;

     responseData.Data.Security = pTask->stSecurity;

     if ((dwError = MessageCreate(&responseData,
          sizeof(responseData),
          &pMessage)) != ERROR_SUCCESS)
     {
          return dwError;
     }

     if ((dwError = MessageArrayAddMessage(pMessageList, pMessage))
!= ERROR_SUCCESS)
     {
          return dwError;
     }


     /* Преобразование действий в сообщения*/
     for (DWORD i = 0; i < pTask->ActionList.dwCount; i++)
     {
          responseData.dataType = REQRES_DATA_ACTION;
          responseData.Data.Action = *(pTask-
>ActionList.pActions[i]);

          if ((dwError = MessageCreate(&responseData,
               sizeof(responseData),
               &pMessage)) != ERROR_SUCCESS)
```

```
          {
               return dwError;
          }

          if ((dwError = MessageArrayAddMessage(pMessageList,
pMessage)) != ERROR_SUCCESS)
          {
               return dwError;
          }
     }


     /* Преобразование триггеров в сообщения*/
     for (DWORD i = 0; i < pTask->stTringgerList.dwCount; i++)
     {
          responseData.dataType = REQRES_DATA_TRIGGER;
          responseData.Data.Trigger = *(pTask-
>stTringgerList.pTriggers[i]);

          if ((dwError = MessageCreate(&responseData,
               sizeof(responseData),
               &pMessage)) != ERROR_SUCCESS)
          {
               return dwError;
          }

          if ((dwError = MessageArrayAddMessage(pMessageList,
pMessage)) != ERROR_SUCCESS)
          {
               return dwError;
          }

     }

     return ERROR_SUCCESS;
}
```

```
//================================================================
================
DWORD
TaskActionsPerform(
     _In_ PSTask pTask
) {

     STARTUPINFO StartUpInfo;
     PROCESS_INFORMATION ProcessInfo;
     HANDLE hToken;
     DWORD dwError = ERROR_SUCCESS;

     memset(&StartUpInfo, 0, sizeof(StartUpInfo));
     memset(&ProcessInfo, 0, sizeof(ProcessInfo));

     StartUpInfo.cb = sizeof(StartUpInfo);


     for (DWORD i = 0; i < pTask->ActionList.dwCount; i++)
     {
          PSAction performAction = pTask->ActionList.pActions[i];

          if (!performAction->isEnable)
               continue;

          switch (performAction->ActionType)
          {

          case ACTION_EXECUTE_PROGRAMM:

               if ((dwError = SecurityGetUserToken(&pTask-
>stSecurity, &hToken)) != ERROR_SUCCESS)
               {
               }
```

```
                if (!CreateProcessAsUser(hToken, NULL,
                     performAction-
>Action.ExecuteProgramm.szProgrammPath,
                     NULL, NULL, FALSE, FALSE, NULL, NULL,
                     &StartUpInfo, &ProcessInfo))
                {
                }

                break;
           default:
                break;
           }

     }

     return ERROR_SUCCESS;
}
```

Приложение В.3

Файл «Time.c»

```
/**
* @file Time.c
* Планировщик задач
*/


#include "stdafx.h"

extern PSTimersContext g_pTimerList = NULL;


DWORD
WaitTimersLock()
{
```

```
     if (g_pTimerList->eStatus == WAITING_TIMERS_RUNNING)
     {
          SetEvent(g_pTimerList->hPauseTimersEvent);

          WaitForSingleObject(g_pTimerList->hTimersPausedEvent,
INFINITE);

     }
     return ERROR_SUCCESS;
}

//================================================================
===============
DWORD
WaitTimersUnLock()
{
     if (g_pTimerList->eStatus == WAITING_TIMERS_PAUSED)
     {
          SetEvent(g_pTimerList->hContinueTimersEvent);
     }
     return ERROR_SUCCESS;
}

//================================================================
===============
VOID
WINAPI
WaitForTimersThread(LPVOID lpParams){

     DWORD dwError = ERROR_SUCCESS;
     DWORD dwTimerIndex = 0;

#ifdef OS
     TocRegistryProcess();
#endif
```

```
    while (TRUE){

        dwError = ERROR_SUCCESS;
        g_pTimerList->eStatus = WAITING_TIMERS_RUNNING;

        /* Ожидание таймеров а также событий на добавление новых
таймеров или прекращение работы */
        dwTimerIndex = WaitForTimers(g_pTimerList->dwCount,
g_pTimerList->phTimers, INFINITE);

        switch (dwTimerIndex)
        {

            // Запрос на остановку ожидания таймеров
        case WAITING_TIMERS_PAUSE:

            g_pTimerList->eStatus = WAITING_TIMERS_PAUSED;

            // Установка события остановки ожидания
            SetEvent(g_pTimerList->hTimersPausedEvent);

            // Ожидание события продолжения ожидания
            WaitForSingleObject(g_pTimerList-
>hContinueTimersEvent, INFINITE);

            g_pTimerList->eStatus = WAITING_TIMERS_RUNNING;

            continue;

            break;

        case WAITING_TIMERS_STOP:

            g_pTimerList->eStatus = WAITING_TIMERS_STOPED;
```

```
                SetEvent(g_pTimerList->hTimersPausedEvent);

                return;

                break;

                // Запрос на выполнение задачи связанной с таймером
            default:
            {
                g_pTimerList->eStatus = WAITING_TIMERS_PERFORM_TASK;

                PSTimerDetails pDetails = NULL;

                SYSTEMTIME currentTime = { 0 };
                FILETIME currentFileTime = { 0 };

                GetSystemTime(&currentTime);
                SystemTimeToFileTime(&currentTime,
&currentFileTime);

                // Получение дополнительной информации связанной с
таймером (задача, триггер)
                pDetails = g_pTimerList->pDetails[dwTimerIndex -
g_pTimerList->RESERVED_HANDLE_SIZE];

                pDetails->pTask->prevExecuteTime = currentFileTime;

                SaveTaskToRegistry(pDetails->pTask);

                // Выполнение действий задачи
                if ((dwError = TaskActionsPerform(pDetails->pTask))
!= ERROR_SUCCESS)
                {

                }
```

```
                // Удаление сработавшего триггера триггера
                if ((dwError =
TimerContextDeleteTimer(dwTimerIndex)) != ERROR_SUCCESS)
                {

                }

                // Установка сработавшего триггера триггера
                if ((dwError = TriggerInstall(pDetails->pTask,
pDetails->pTrigger)) != ERROR_SUCCESS)
                {

                }

                // Освобождение памяти из под вспомогательной
структуры связанной с таймером
                free(pDetails);

                break;

            }
            }
        }
}
```

Приложение В.4

Файл «Action.c»

```
/**
* @file Action.c
* Планировщик задач
*/

#include "stdafx.h"
```

```
DWORD
ActionCreate(
_In_ PSAction pActionSrc,
_Out_ PSAction* pAction
)
{
     *pAction = malloc(sizeof(SAction));

     if (*pAction == NULL)
     {
          return ERROR_NOT_ENOUGH_MEMORY;
     }

     memset(*pAction, 0, sizeof(SAction));
     memcpy(*pAction, pActionSrc, sizeof(SAction));

     return ERROR_SUCCESS;
}

DWORD
ActionAddToList(
_Inout_ PSActionList stActionList,
_In_ PSAction pAction
)
{
     if (stActionList->dwCapacity - 1 == stActionList->dwCount)
     {
          stActionList->dwCapacity += 5;
          stActionList->pActions = realloc(stActionList->pActions,
stActionList->dwCapacity * sizeof(PSAction));

          if (stActionList->pActions == NULL)
               return ERROR_NOT_ENOUGH_MEMORY;
     }
```

```
	stActionList->pActions[stActionList->dwCount] = pAction;

	stActionList->dwCount++;

	return ERROR_SUCCESS;
}

DWORD
ActionListInitialize(
_Out_ PSActionList pActionList
)
{
	pActionList->dwCount = 0;
	pActionList->dwCapacity = 5;

	pActionList->pActions = malloc(pActionList->dwCapacity *
sizeof(PSAction));

	if (pActionList->pActions == NULL)
		return ERROR_NOT_ENOUGH_MEMORY;

	return ERROR_SUCCESS;
}
```

Приложение В.5

Файл « RequestHandler.c»

```
/**
* @file RequestHandler.c
* Планировщик задач
*/

#include "stdafx.h"
```

```
extern HANDLE* g_pTimersHandles = NULL; // Указатель на массив
описателей таймеров

VOID
WINAPI
ClientRequestReciver(
     LPVOID lpParam
)
{
     DWORD dwWaitStopReason = 0;
     DWORD dwBytesRead = 0;
     DWORD dwError = ERROR_SUCCESS;

     SReqResInfo RequestInfo = { 0 };
     BOOL fFirstMessage = TRUE;

     BYTE cRequest[PIPE_BUFFER_SIZE];

     HANDLE hPipe = INVALID_HANDLE_VALUE;

     // Структурами для перекрывающего подключения
     OVERLAPPED overlappedConnection = { 0 };

     // Событие подключения нового клиента
     HANDLE connectedEvent = CreateEvent(
          NULL,
          TRUE,
          TRUE,
          NULL);

     overlappedConnection.hEvent = connectedEvent;

     if (GetPipeInstance(&hPipe) != ERROR_SUCCESS) {
          return;
     }
```

```
     HANDLE waitInterruptEvents[2] = {
          g_ServiceStopEvent,
          connectedEvent
     };

     __try
     {
          /* Цикл обработки запросов */
          while (TRUE)
          {

               dwWaitStopReason = 0;
               fFirstMessage = TRUE;
               dwError = ERROR_SUCCESS;
               SMessageArray MessageList = { 0 };

               __try
               {
                    ConnectToNewClient(hPipe,
&overlappedConnection);

                    dwWaitStopReason = WaitForMultipleObjects(
                         SIZE_OF_ARRAY(waitInterruptEvents),
                         waitInterruptEvents, FALSE, INFINITE);

                    if (dwWaitStopReason == 0)
                         return 0;

                    RequestInfo.dwMessageCount = 1;

                    // Список сообщений от клиента
                    if ((dwError =
MessageArrayCreate(&MessageList)) != ERROR_SUCCESS) {
                         continue;
                    }
```

```
                    /* Чтение всех сообщений из канала */
                    while (fFirstMessage ||
RequestInfo.dwMessageCount)
                    {
                         /* Чтение из именованного канала */
                         if (!ReadFile(hPipe, cRequest,
PIPE_BUFFER_SIZE, &dwBytesRead, NULL))
                         {
                              dwError = GetLastError();
                              break;
                         }

                         if (fFirstMessage)
                         {
                              memcpy(&RequestInfo, &cRequest,
sizeof(RequestInfo));
                              fFirstMessage = FALSE;
                         }
                         else
                         {
                              PSMessage pMessage = NULL;

                              if ((dwError =
MessageCreate(cRequest, dwBytesRead, &pMessage)) != ERROR_SUCCESS)
                              {
                                   break;
                              }

                              if ((dwError =
MessageArrayAddMessage(&MessageList, pMessage)) != ERROR_SUCCESS)
                              {
                                   break;
                              }
                         }

                         RequestInfo.dwMessageCount--;
```

```
                    }

                    /* Проверка ошибок при чтении */
                    if (dwError != ERROR_SUCCESS)
                    {
                         continue;
                    }

                    /* Задача */
                    PSTask pTask = NULL;

                    /* Создание задачи */
                    TaskCreate(&pTask);

                    __try
                    {
                         /* Получение данных задачи из полученных
сообщений */
                         if ((dwError = ParseMessages(&MessageList,
pTask)) != ERROR_SUCCESS)
                         {
                              continue;
                         }

                         /* Выполнение запрашиваемой задачи */
                         if ((dwError = PerformRequest(hPipe,
&RequestInfo, pTask)) != ERROR_SUCCESS)
                         {
                              SendErrorToClient(hPipe, dwError);
                              continue;
                         }
                    }
                    __finally
                    {
                         if (AbnormalTermination())
                         {
```

```
                              TaskPlannerFree(pTask);
                         }
                    }
               }
               __finally
               {
                    // Отключение клиента от экземпляра канала
                    DisconnectNamedPipe(hPipe);
                    MessageArrayFreeAllMessages(&MessageList);
               }
          }
     }
     __finally
     {
          if (hPipe != INVALID_HANDLE_VALUE)
          {
               CloseHandle(hPipe);
          }
     }
}

//=============================================================
==============
DWORD
SendErrorToClient(
     _In_ HANDLE hPipe,
     _In_ DWORD dwError)
{

     DWORD dwWritten = 0;
     SReqResInfo responseInfo = { 0 };
     SReqResData responseDate = { 0 };

     /* Служебная информация (кол-во сообщений) */
     responseInfo.dwMessageCount = 1;
     responseInfo.Details.ErrorInfo.dwErrorCode = dwError;
```

```
	responseDate.dataType = REQRES_DATA_INFO;
	responseDate.Data.RequestInfo = responseInfo;
	responseDate.Data.RequestInfo.RequstInfoType =
RESPONSE_INFO_ERROR;

	/* Отправка служебной информации */
	WriteFile(hPipe, &responseDate, sizeof(responseDate),
&dwWritten, NULL);
	FlushFileBuffers(hPipe);

	responseInfo.dwMessageCount = 0;

	/* Отправка служебной информации */
	WriteFile(hPipe, &responseDate, sizeof(responseDate),
&dwWritten, NULL);
	FlushFileBuffers(hPipe);


	return ERROR_SUCCESS;
}

//=============================================================
===============
BOOL
ConnectToNewClient(
	_In_ HANDLE hPipe,
	_Inout_ LPOVERLAPPED lpo
)
{
	BOOL fConnected, fPendingIO = FALSE;

	fConnected = ConnectNamedPipe(hPipe, lpo);

	if (fConnected)
	{
```

```
            return 0;
        }

        switch (GetLastError())
        {
            // Соединение в процессе
        case ERROR_IO_PENDING:
            fPendingIO = TRUE;
            break;

            // Клиент уже подключен
        case ERROR_PIPE_CONNECTED:
            if (SetEvent(lpo->hEvent))
                break;

        default:
            return 0;

        }

        return fPendingIO;
}

//=================================================================
================
DWORD
WINAPI
ParseMessages(
        _In_ PSMessageArray pMessageList,
        _Out_ PSTask pTask
)
{
        DWORD dwError = ERROR_SUCCESS;

        for (DWORD i = 0; i < pMessageList->dwCount; i++)
        {
```

```
            PSReqResData pMessage = pMessageList->pMessages[i]-
>pData;

            switch (pMessage->dataType)
            {
            case REQRES_DATA_TASK:

                  pTask->isEnable = pMessage-
>Data.GeneralTask.isEnable;

                  memcpy(pTask->szTaskName, pMessage-
>Data.GeneralTask.szTaskName, sizeof(pTask->szTaskName));
                  memcpy(pTask->szDescription, pMessage-
>Data.GeneralTask.szDescription, sizeof(pTask->szDescription));

                  pTask->nextExecuteTime = pMessage-
>Data.GeneralTask.nextExecuteTime;
                  pTask->prevExecuteTime = pMessage-
>Data.GeneralTask.prevExecuteTime;

                  break;

            case REQRES_DATA_SECURITY:

                  memcpy(pTask->stSecurity.szUserName, pMessage-
>Data.Security.szUserName, sizeof(pTask->stSecurity.szUserName));
                  memcpy(pTask->stSecurity.szUserPassword, pMessage-
>Data.Security.szUserPassword, sizeof(pTask-
>stSecurity.szUserPassword));

                  break;

            case REQRES_DATA_SETTINGS:
```

```
                pTask->stSettings.DeleteDisabledTaskIn = pMessage-
>Data.Setting.DeleteDisabledTaskIn;
                pTask->stSettings.TerminateExecutedProcessIn =
pMessage->Data.Setting.TerminateExecutedProcessIn;

                break;

           case REQRES_DATA_ACTION:
           {
                PSAction pAction = NULL;

                if (pTask->ActionList.dwCapacity == 0)
                {
                     if ((dwError = ActionListInitialize(&pTask-
>ActionList)) != ERROR_SUCCESS) {
                          return dwError;
                     }
                }

                if ((dwError = ActionCreate((PSAction)&pMessage-
>Data, &pAction)) != ERROR_SUCCESS) {
                     return dwError;
                }

                if ((dwError = ActionAddToList(&pTask->ActionList,
pAction)) != ERROR_SUCCESS) {
                     return dwError;
                }

                break;
           }
           case REQRES_DATA_TRIGGER:
           {
                PSTrigger pTrigger = NULL;

                if (pTask->stTringgerList.dwCapacity == 0)
```

```
                {
                     if ((dwError = TriggerListInitialze(&pTask-
>stTringgerList)) != ERROR_SUCCESS)
                     {
                          return dwError;
                     }
                }

                if ((dwError = TriggerCreate((PSTrigger)&pMessage-
>Data, &pTrigger)) != ERROR_SUCCESS)
                {
                     return dwError;
                }

                if ((dwError = TriggerListAddTrigger(&pTask-
>stTringgerList, pTrigger)) != ERROR_SUCCESS)
                {
                     return dwError;
                }

                break;
           }
           default:

                return ERROR_REQUST_UNDEFINED;
                break;

           }
     }

     return ERROR_SUCCESS;
}

//=================================================================
================
DWORD
```

```
MessageArrayCreate(
     _Out_ PSMessageArray pMessageList
)
{
     pMessageList->dwCount = 0;
     pMessageList->dwCapacity = 5;

     pMessageList->pMessages = malloc(pMessageList->dwCapacity *
sizeof(PSMessage));

     if (pMessageList->pMessages == NULL)
          return ERROR_NOT_ENOUGH_MEMORY;

     return ERROR_SUCCESS;
}

//================================================================
===============
DWORD
MessageCreate(
     _In_ PVOID pData,
     _In_ DWORD dwSize,
     _Out_ PSMessage* pMessage
)
{
     (*pMessage) = malloc(sizeof(SMessage));

     if ((*pMessage) == NULL)
          return ERROR_NOT_ENOUGH_MEMORY;

     (*pMessage)->dwSize = dwSize;
     (*pMessage)->pData = malloc(dwSize);

     if ((*pMessage)->pData == NULL) {

          free(*pMessage);
```

```
            return ERROR_NOT_ENOUGH_MEMORY;

        }

        memcpy((*pMessage)->pData, pData, dwSize);

        return ERROR_SUCCESS;

}

//=================================================================
================
DWORD
MessageFree(
        _In_ PSMessage pMessage
)
{
        free(pMessage->pData);
        free(pMessage);

        return ERROR_SUCCESS;

}

//=================================================================
================
DWORD
MessageArrayAddMessage(
        _In_ PSMessageArray pMessageList,
        _In_ PSMessage pMessage
)
{

        if (pMessageList->dwCapacity - 1 == pMessageList->dwCount)
        {
            pMessageList->dwCapacity += 5;
```

```
          pMessageList->pMessages = realloc(pMessageList-
>pMessages, pMessageList->dwCapacity * sizeof(PSMessage));

          if (pMessageList->pMessages == NULL)
               return ERROR_NOT_ENOUGH_MEMORY;
     }

     pMessageList->pMessages[pMessageList->dwCount] = pMessage;

     pMessageList->dwCount++;

     return ERROR_SUCCESS;

}

//=================================================================
================
DWORD
MessageArrayDeleteMessage(
     _In_ PSMessageArray pMessageList,
     _In_ PSMessage pMessage
)
{
     DWORD dwIndex = 0;

     /* Поиск позиции сообщения*/
     for (DWORD i = 0; i < pMessageList->dwCount; i++)
     {
          if (pMessageList->pMessages[i] == pMessage)
          {
               dwIndex = i;
               break;
          }
     }

     /* Замещение задачами*/
```

```
	for (DWORD i = dwIndex; i < pMessageList->dwCount; i++)
	{
		pMessageList->pMessages[i] = pMessageList->pMessages[i +
1];
	}

	pMessageList->dwCount--;

	return ERROR_SUCCESS;

}

//================================================================
===============
DWORD
MessageArrayFreeAllMessages(
	_In_ PSMessageArray pMessageList
)
{
	for (DWORD i = 0; i < pMessageList->dwCount; i++)
	{
		MessageFree(pMessageList->pMessages[i]);
	}

	free(pMessageList->pMessages);

	return ERROR_SUCCESS;
}

//================================================================
===============
DWORD
PerformRequest(
	_In_ HANDLE hPipe,
	_In_ PSReqResInfo pReqInfo,
	_In_ PSTask pTask
```

```
)
{
	DWORD dwWritten = 0;
	DWORD dwError = ERROR_SUCCESS;
	SMessageArray messageList = { 0 };
	SReqResInfo responseInfo = { 0 };
	SReqResData responseDate = { 0 };

	__try
	{
		WaitTimersLock();

		switch (pReqInfo->RequstInfoType)
		{
		case REQRES_INFO_ADD:
		{

			if ((dwError = TaskAdd(pTask)) != ERROR_SUCCESS)
			{
				return dwError;
			}

			if ((dwError = MessageArrayCreate(&messageList))
				!= ERROR_SUCCESS)
			{
				return dwError;
			}

			__try
			{
				/* Преобразование задачи в список сообщений */
				if ((dwError = TaskConvertToMessages(pTask,
&messageList))
					!= ERROR_SUCCESS)
				{
					return dwError;
```

```
                    }

                    /* Служебная информация (кол-во сообщений) */
                    responseInfo.dwMessageCount =
(BYTE)messageList.dwCount + 1;
                    responseDate.dataType = REQRES_DATA_INFO;
                    responseDate.Data.RequestInfo = responseInfo;

                    /* Отправка служебной информации */
                    WriteFile(hPipe, &responseDate,
sizeof(responseDate), &dwWritten, NULL);
                    FlushFileBuffers(hPipe);

                    /* Отправка сообщений */
                    for (DWORD i = 0; i < messageList.dwCount; i++)
                    {
                         WriteFile(hPipe, messageList.pMessages[i]-
>pData, messageList.pMessages[i]->dwSize, &dwWritten, NULL);
                         FlushFileBuffers(hPipe);
                    }

               }
               __finally
               {
                    MessageArrayFreeAllMessages(&messageList);
               }
               break;

          }
          case REQRES_INFO_EDIT:

               if ((dwError = TaskEdit(pTask)) != ERROR_SUCCESS)
               {
                    return dwError;

               }
```

```
                break;

            case REQRES_INFO_DELETE:

                if ((dwError = TaskDelete(pTask)) != ERROR_SUCCESS)
                {
                    return dwError;
                }

                break;

            case REQRES_INFO_GETTASKS:
            {
                DWORD dwError = ERROR_SUCCESS;
                PSLinkedHashTableEntry pTaskEntry = NULL;

                LinkedHashTableGetFirstEntry(g_pTaskHashTable,
&pTaskEntry);


                while (pTaskEntry != NULL)
                {

                    if ((dwError =
MessageArrayCreate(&messageList))
                            != ERROR_SUCCESS)
                    {
                        return dwError;
                    }


                    /* Преобразование задачи в список сообщений */
                    if ((dwError =
TaskConvertToMessages(pTaskEntry->pGenericElem->pData,
&messageList))
                            != ERROR_SUCCESS)
```

```
                    {
                         return dwError;
                    }

                    /* Служебная информация (кол-во сообщений) */
                    responseInfo.dwMessageCount =
(BYTE)messageList.dwCount + 2;
                    responseDate.dataType = REQRES_DATA_INFO;
                    responseDate.Data.RequestInfo = responseInfo;

                    /* Отправка служебной информации */
                    WriteFile(hPipe, &responseDate,
sizeof(responseDate), &dwWritten, NULL);
                    FlushFileBuffers(hPipe);

                    /* Отправка сообщений */
                    for (DWORD i = 0; i < messageList.dwCount; i++)
                    {
                         WriteFile(hPipe, messageList.pMessages[i]-
>pData, messageList.pMessages[i]->dwSize, &dwWritten, NULL);
                         FlushFileBuffers(hPipe);
                    }

                    MessageArrayFreeAllMessages(&messageList);

                    LinkedHashTableGetNextEntry(g_pTaskHashTable,
pTaskEntry, &pTaskEntry);
               }

               /* последнего сообщения */
               responseInfo.dwMessageCount = 0;
               responseDate.dataType = REQRES_DATA_INFO;
               responseDate.Data.RequestInfo = responseInfo;

               /* Отправка последнего информации */
```

```
                WriteFile(hPipe, &responseDate,
sizeof(responseDate), &dwWritten, NULL);
                FlushFileBuffers(hPipe);

                break;
           }

           default:
                break;
           }

     }
     __finally
     {
           WaitTimersUnLock();
     }

     return ERROR_SUCCESS;
}

//=============================================================
===============
DWORD
GetPipeInstance(
     _Out_ PHANDLE phPipe
)
{
     DWORD dwError = ERROR_SUCCESS;

     // Для доступа к именованном каналу из клиентского приложения
     PACL pDacl = NULL;
     SECURITY_ATTRIBUTES SecAttribs = { 0 };
     SECURITY_DESCRIPTOR SecurityDescriptor = { 0 };

     /* Создание дескриптора безопасности для доступа клиентов к
pipe */
```

```
    InitializeSecurityDescriptor(&SecurityDescriptor,
SECURITY_DESCRIPTOR_REVISION);
    SetSecurityDescriptorDacl(&SecurityDescriptor, TRUE, NULL,
FALSE);

    SecAttribs.nLength = sizeof(SECURITY_ATTRIBUTES);
    SecAttribs.lpSecurityDescriptor = &SecurityDescriptor;

    /* Создание именованного канала */
    (*phPipe) = CreateNamedPipe(PIPE_NAME, PIPE_ACCESS_DUPLEX |
FILE_FLAG_OVERLAPPED,
         PIPE_TYPE_MESSAGE | PIPE_WAIT, PIPE_UNLIMITED_INSTANCES,
         PIPE_BUFFER_SIZE, PIPE_BUFFER_SIZE, 5000, &SecAttribs);

    if ((*phPipe) == INVALID_HANDLE_VALUE)
    {
         dwError = GetLastError();
         return dwError;
    }

    return ERROR_SUCCESS;
}
```

Приложение С

Результаты работы программы

Приложение С.1 – Вид начально экрана

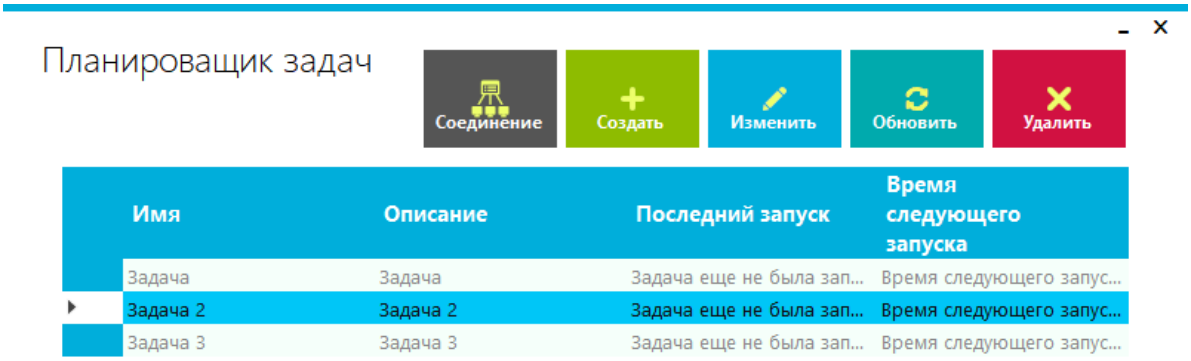


Рисунок 24 - Вид начально экрана

Приложение С.2 – Вид экрана создания задачи

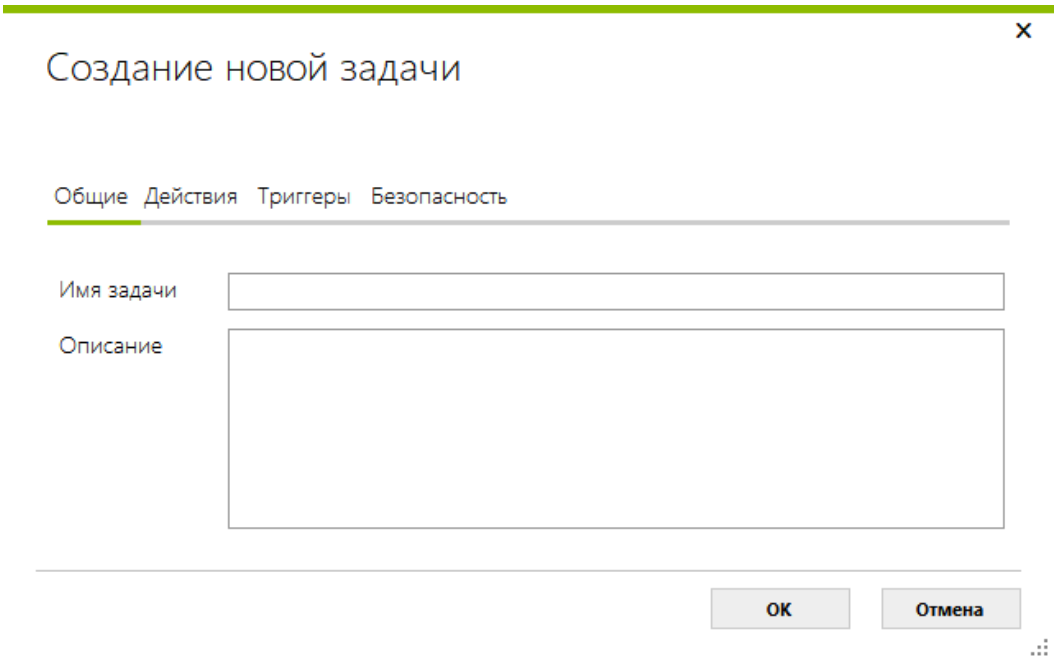


Рисунок 25 - Вид экрана создания задачи

Приложение С.3 – Вид экрана с настройками действия

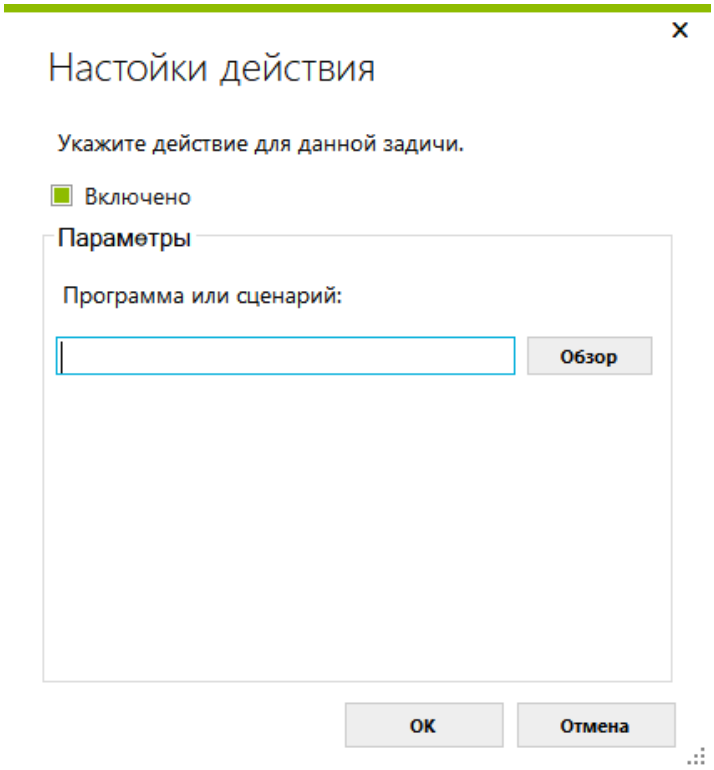


Рисунок 26 - Вид экрана с настройками действия

Приложение С.4 – Вид экрана настроек триггера

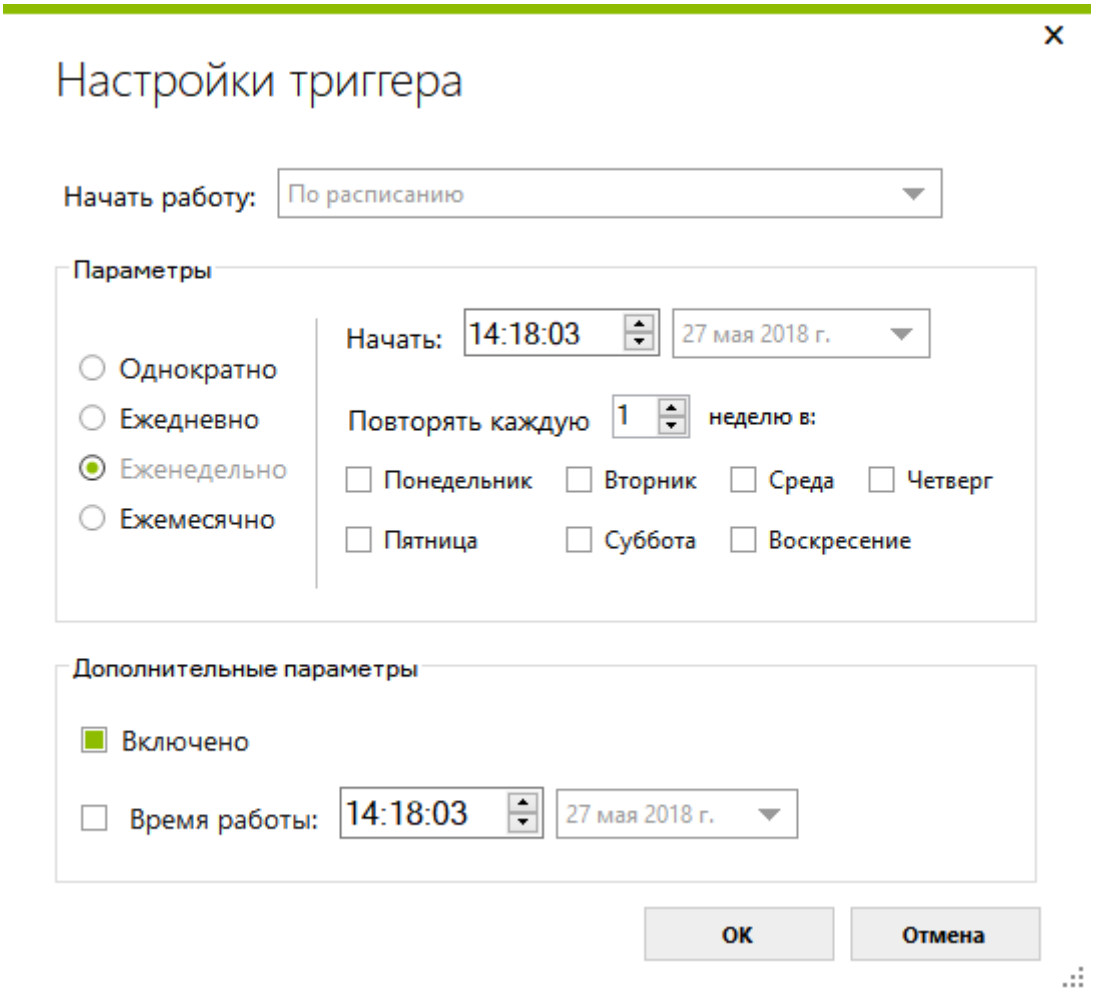


Рисунок 27 – Вид экрана с настройками триггера

Приложение С.5 – Вид экрана подключения к серверу

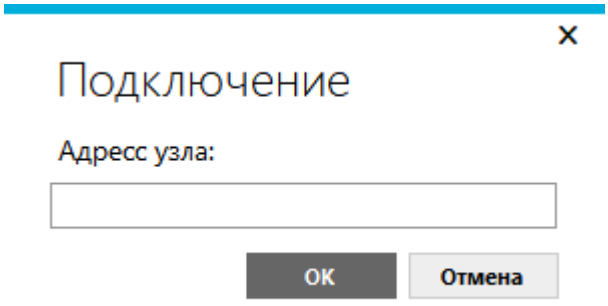


Рисунок 28 – Вид экрана подключения к серверу

Приложение С.6 – Вид экрана редактирования задачи

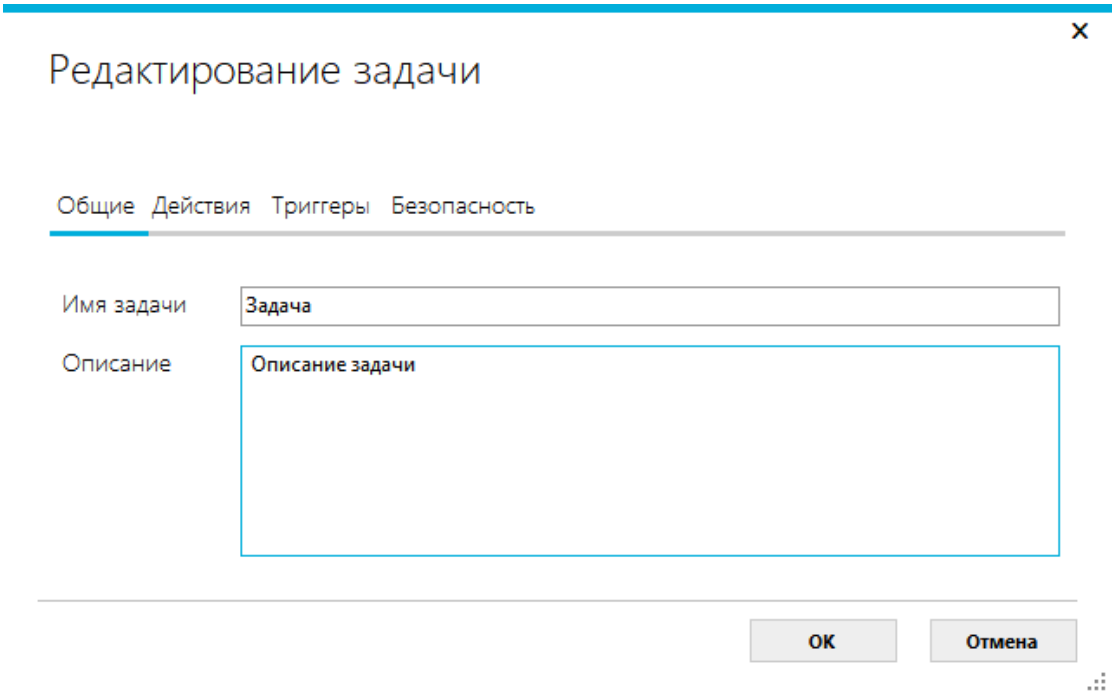


Рисунок 29 – Вид экрана редактирования задачи

Приложение С.7 – Вид окна с информационным сообщением

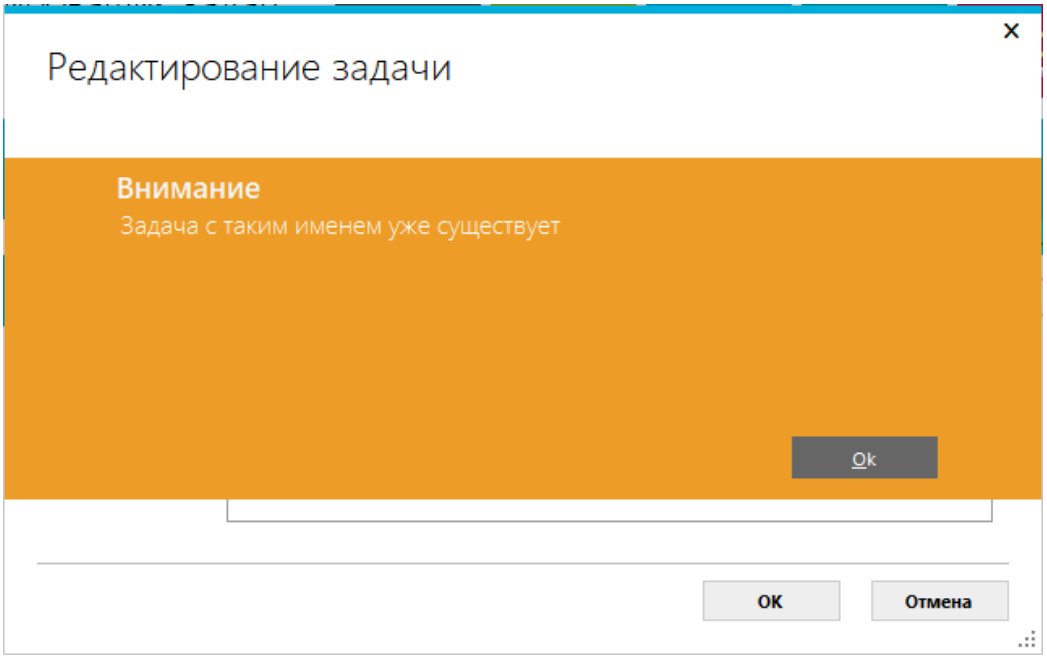


Рисунок 30 – Вид окна с информационным сообщением

Приложение D

UML-диаграммы приложения

Приложение D.1 – UML-диаграмма классов клиента

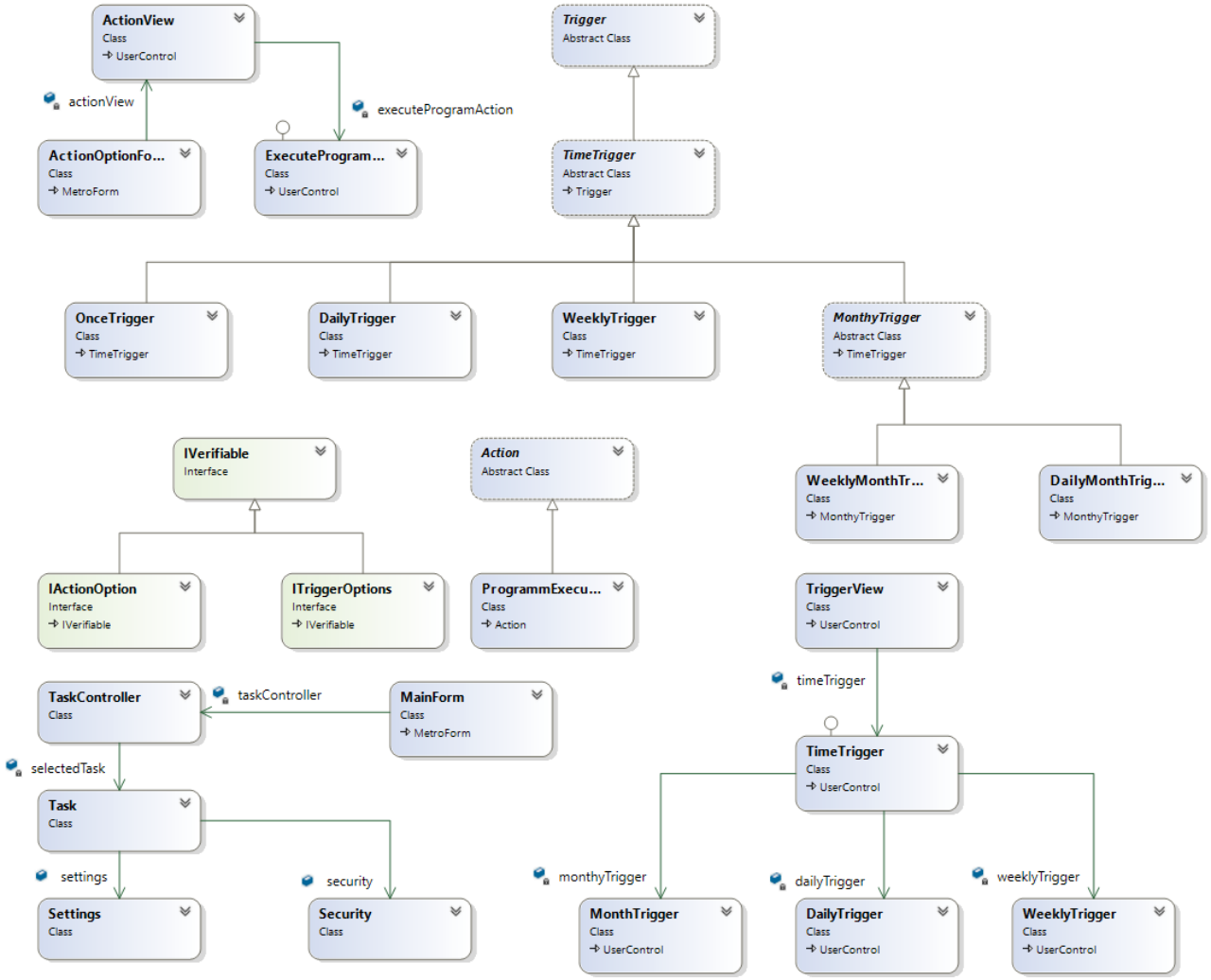


Рисунок 31 – UML-диаграмма классов клиента

Приложение D.2 – UML-диаграмма вариантов использования сервера

Рисунок 32 – UML-диаграмма вариантов использования сервера

Приложение D.3 – UML-диаграмма вариантов использования клиента

Рисунок 33 – UML-диаграмма вариантов использования клиента

Приложение D.4 – UML-диаграмма деятельности сервера

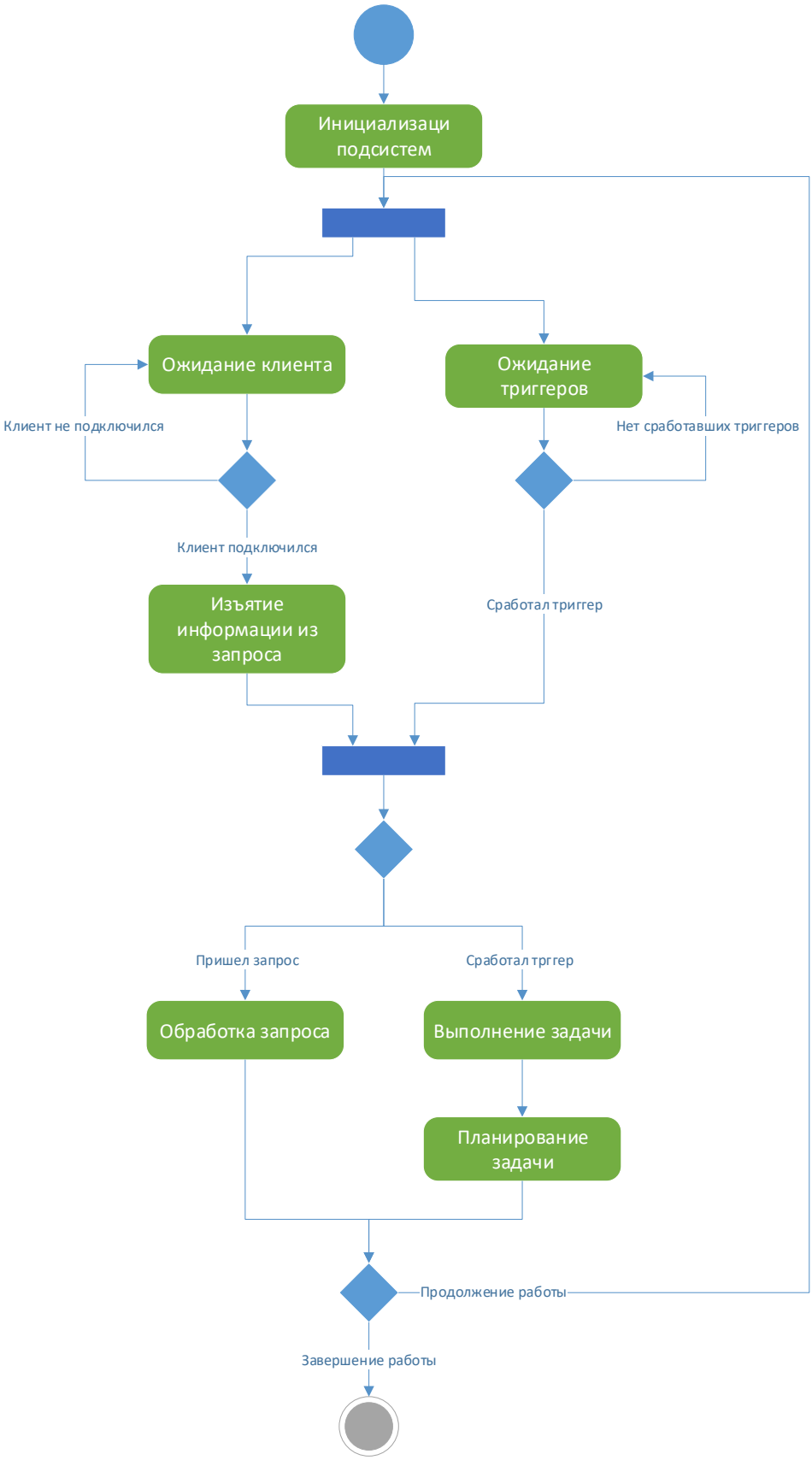


Рисунок 34 – UML-диаграмма деятельности сервера

Приложение D.5 – UML-диаграмма деятельности клиента

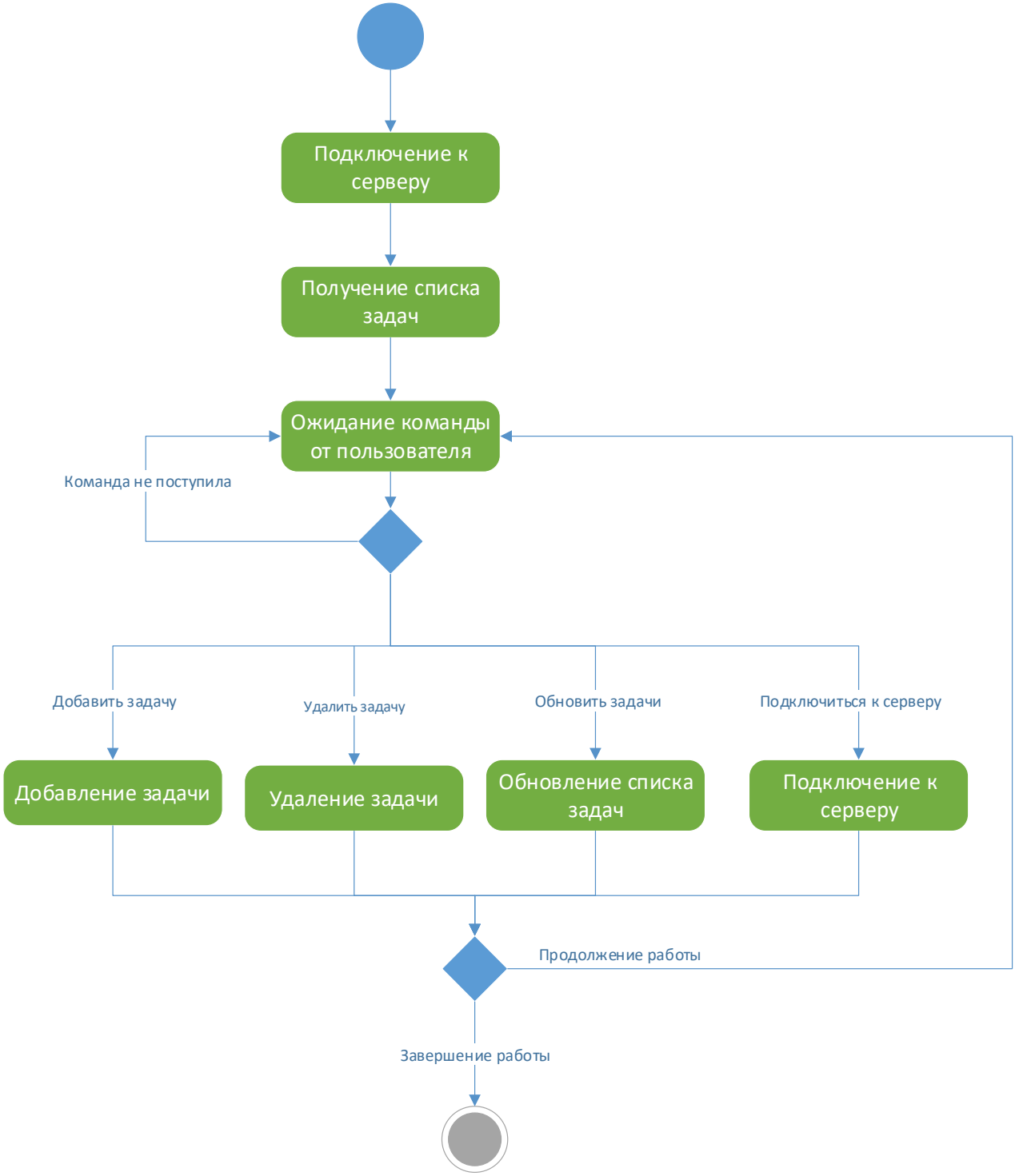


Рисунок 35 – UML-диаграмма деятельности клиента

Приложение D.6 – UML-диаграмма последовательности

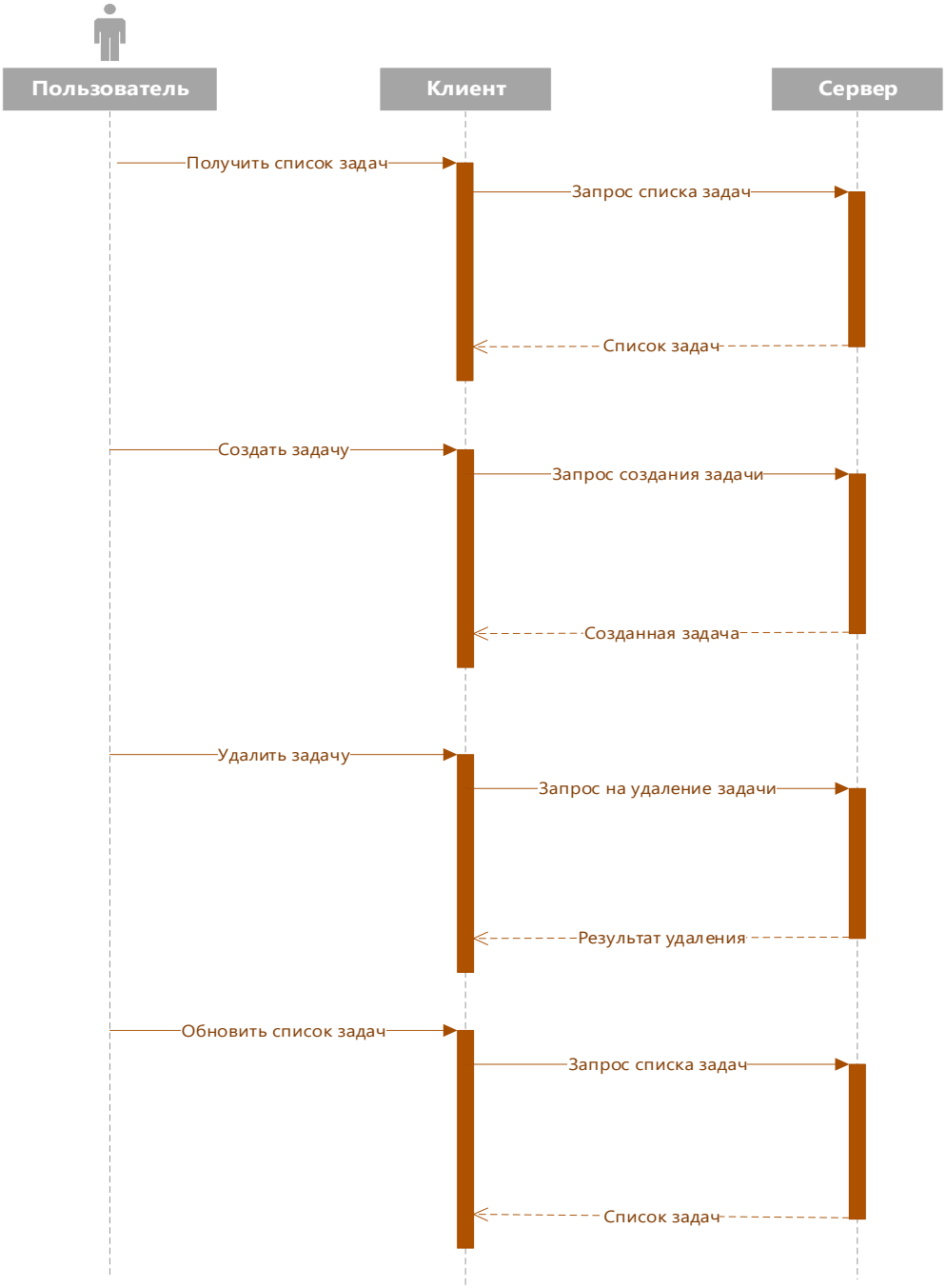


Рисунок 36 – UML-диаграмма последовательности

Приложение D.7 – UML-диаграмма развертывания

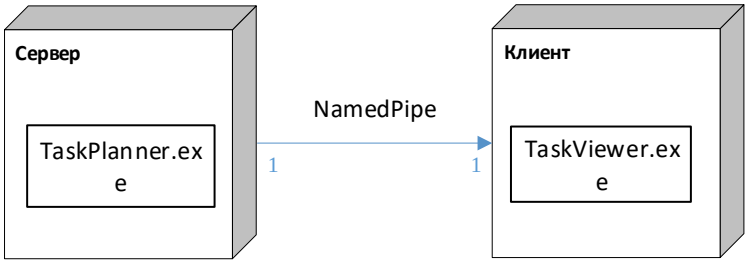


Рисунок 37 – UML-диаграмма развертывания

Приложение Е

Презентация

Постановка задачи

Требуется разработать клиент-серверное приложение, позволяющее планировать выполнение задач на удаленных и локальных компьютерах по расписанию.

- Клиент должен подключаться к удаленным или локальным ЭВМ и осуществлять управление задачами;
- Сервер должен обрабатывать запросы клиента. Осуществлять планирование и выполнение задач;
- Сетевой интерфейс между компьютерами должен быть реализован с помощью именованных каналов.

Диаграмма последовательности

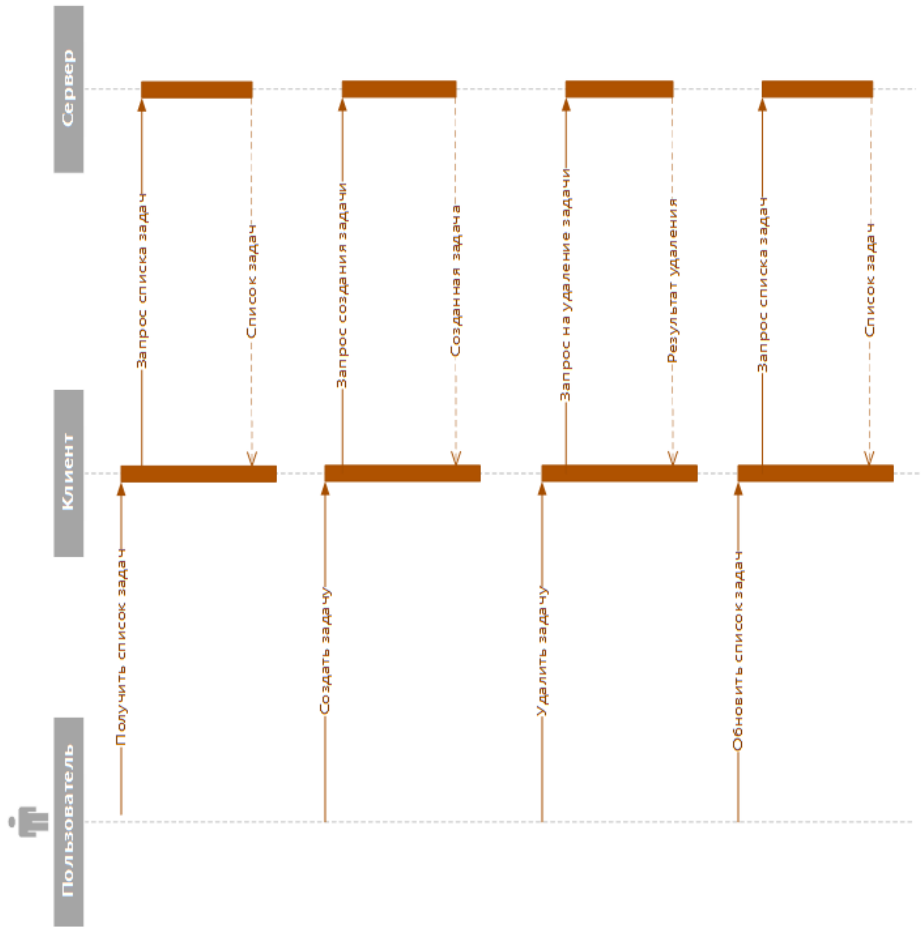


Диаграмма вариантов использования клиента

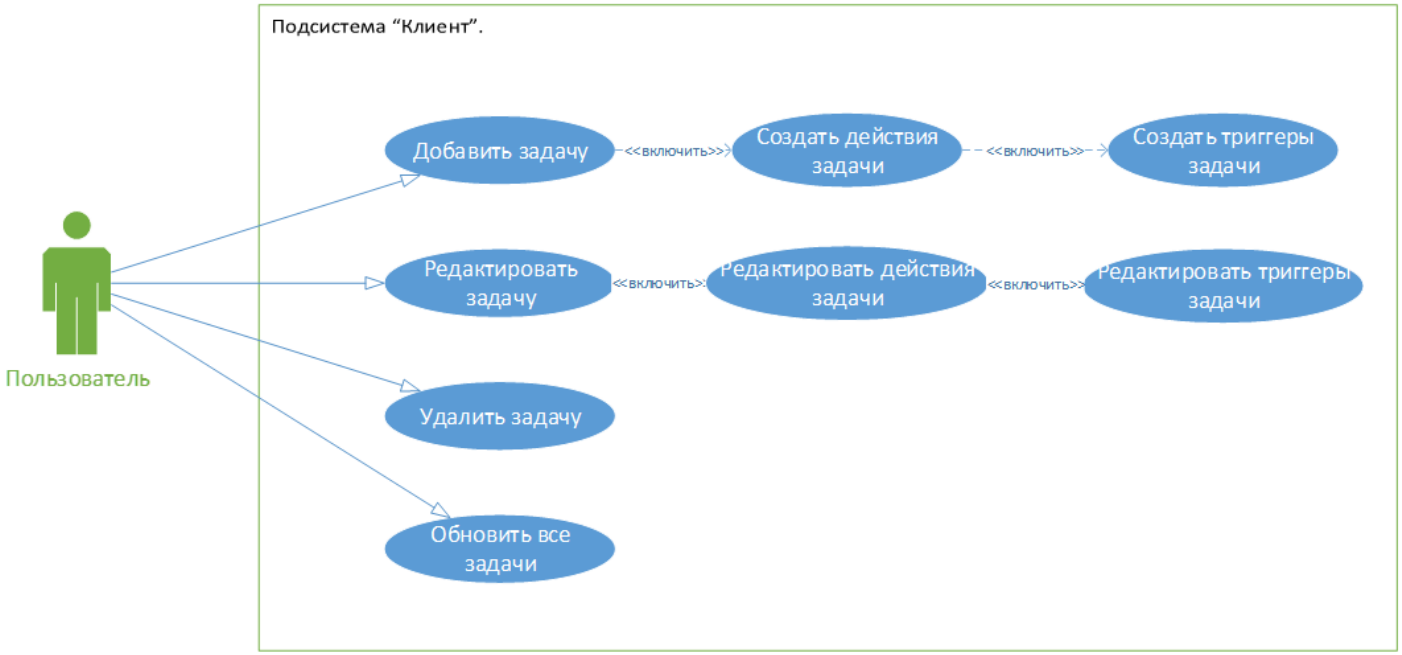


Диаграмма вариантов использования сервера

Диаграмма деятельности клиента

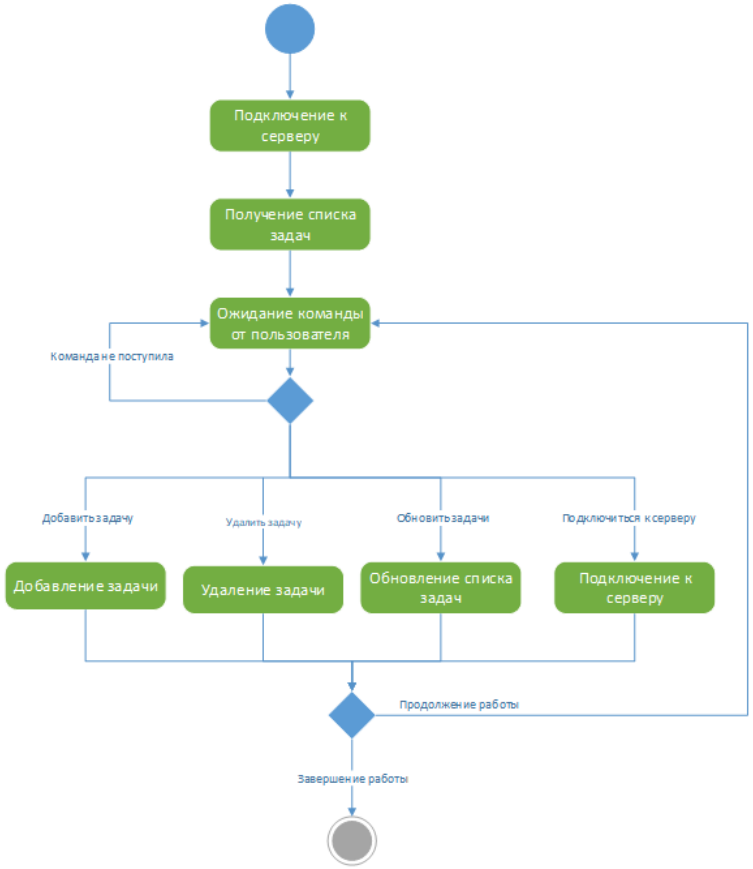


Диаграмма деятельности сервера

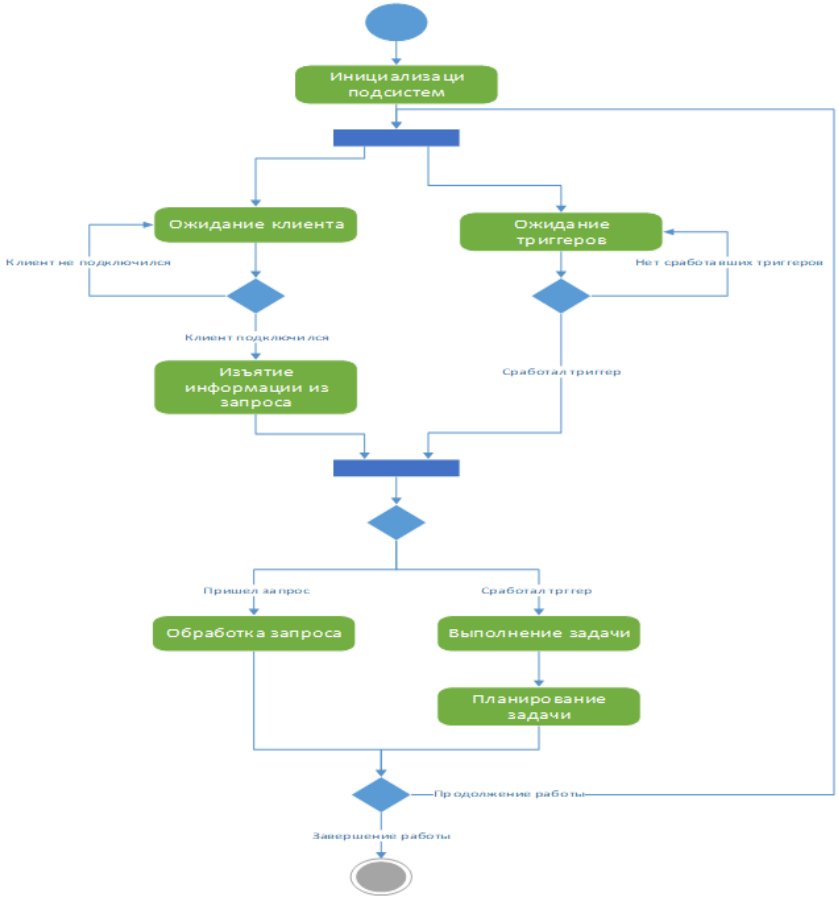
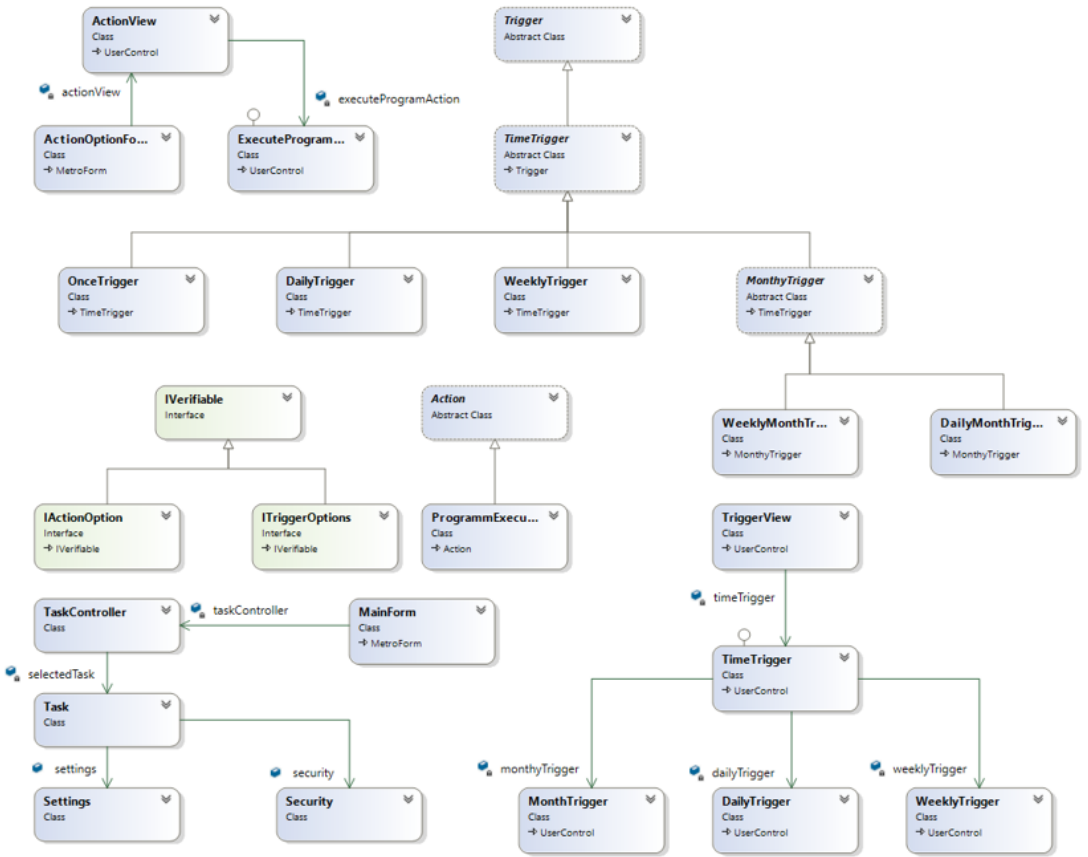
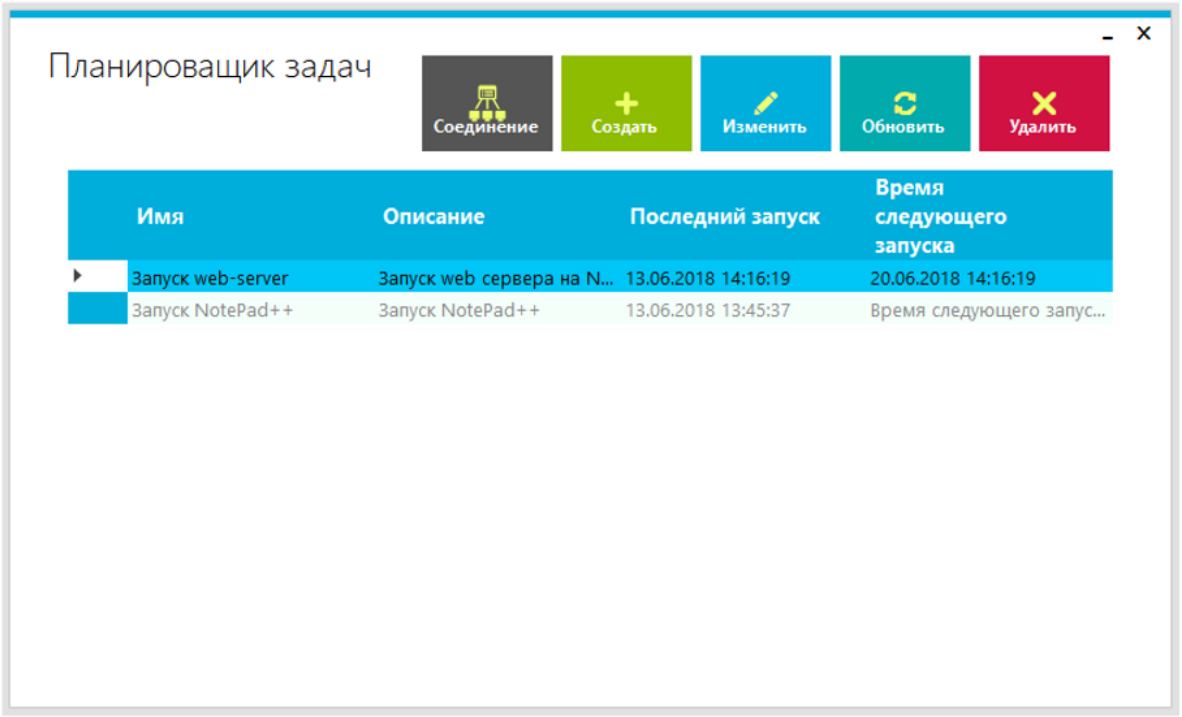


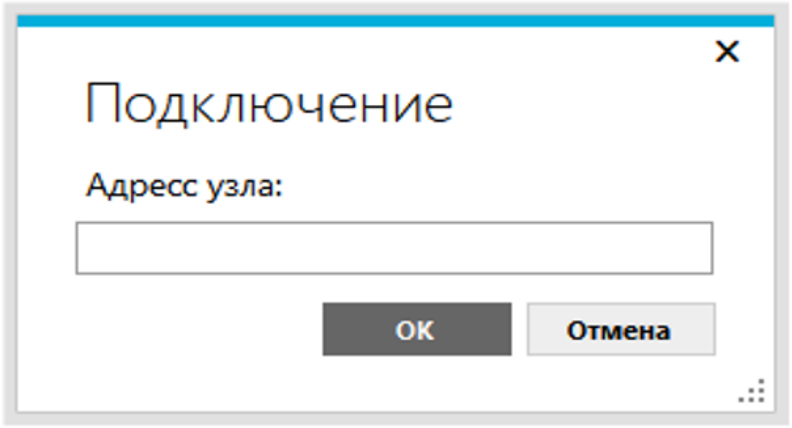
Диаграмма классов клиента



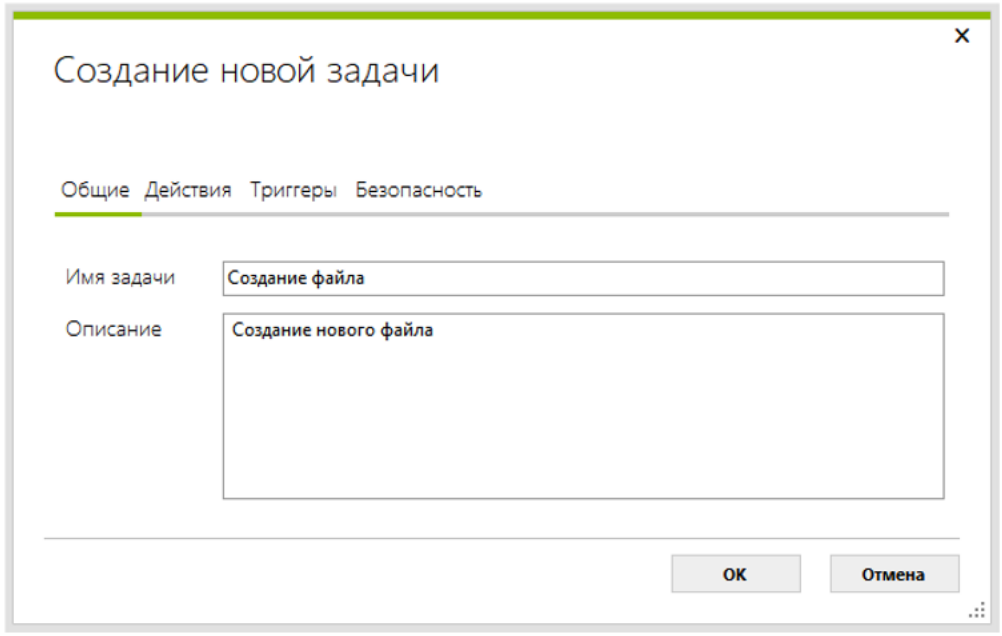
Интерфейс клиентского приложения



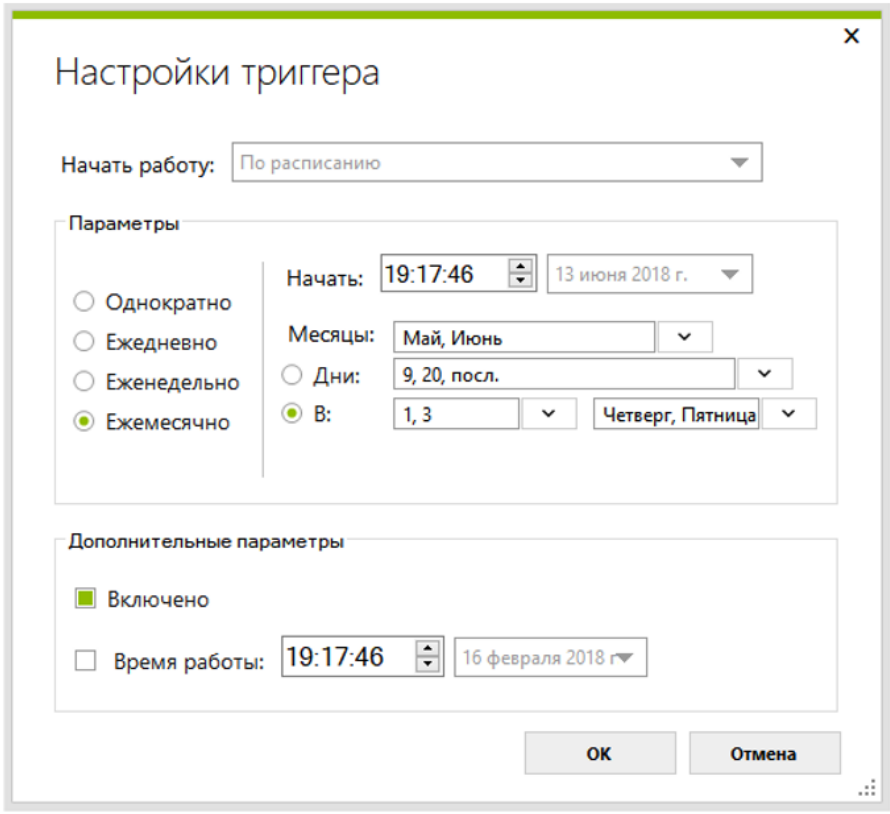
Подключение к серверу



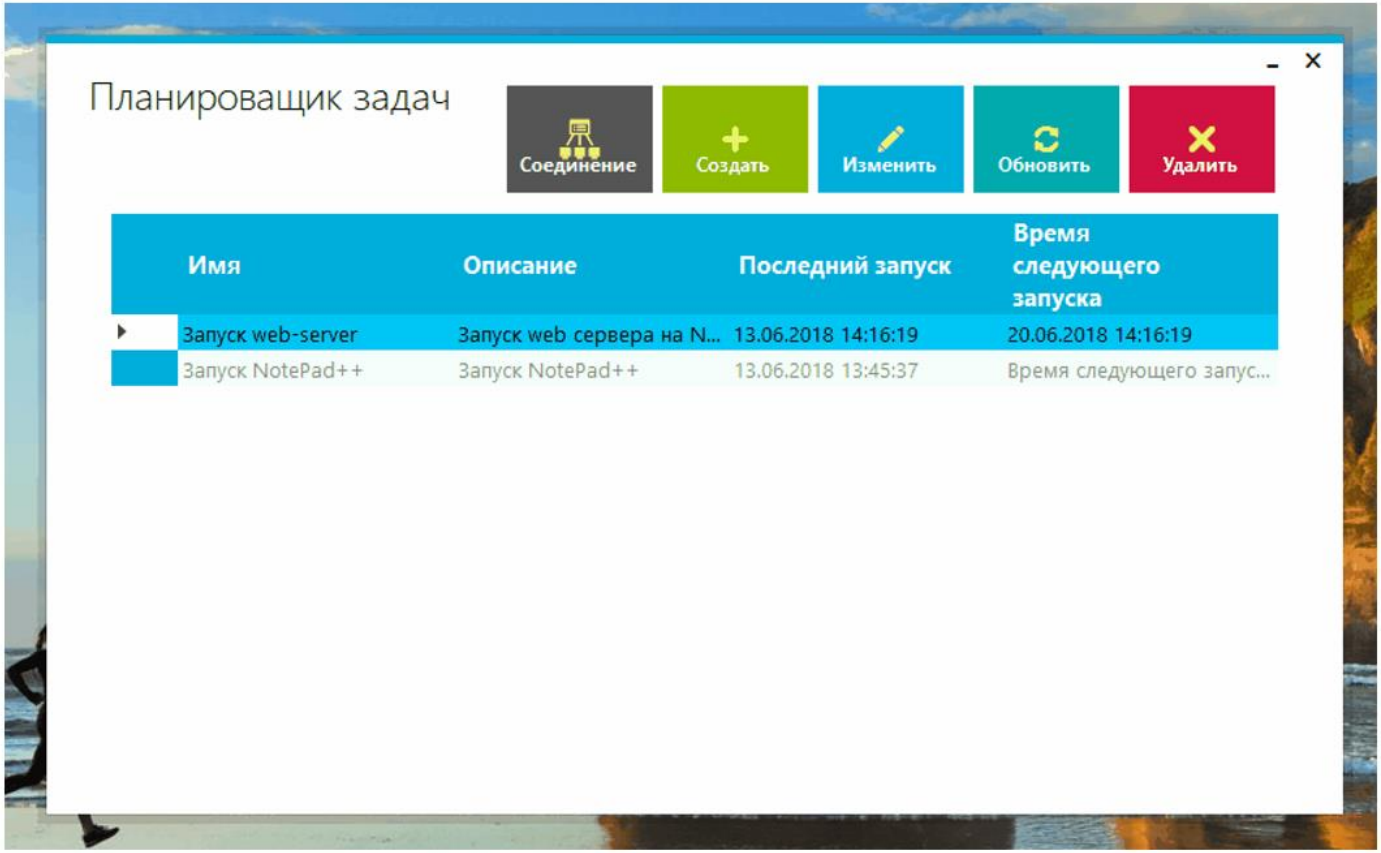
Создание задачи на клиенте



Настройки триггера



Демонстрация работы программы



Заключение

В результате создан проект удовлетворяющий поставленным задачам.

- Клиент может управлять задачами как на локальных, так и удаленных компьютерах;
- Сервер осуществляет планирование, выполнение задач. А также обрабатывает запросы клиента.
- Взаимодействие клиента и сервера может осуществляться по локальной или глобальной сети.

Спасибо за внимание!