Санкт-Петербургский политехнический университет Петра Великого

Институт компьютерных наук и технологий

Кафедра информационных и управляющих систем

|  |
| --- |
| Работа допущена к защите  Зав. кафедрой  И.Г. Черноруцкий  « » 2015 г. |

**ВЫПУСКНАЯ РАБОТА БАКАЛАВРА**

Тема: ***Разработка приложения для передачи данных в условиях нестабильного сетевого соединения с учетом их приоритетов***

**Направление:** 09.03.04 – Программная инженерия

Выполнил студент гр. 43504/1 Усманов М.Д.

Руководитель Коликова Т. В.

Санкт-Петербург

2015

**ЗАДАНИЕ**

**к работе на соискание степени бакалавра**

**студенту** Усманову Максиму Далеровичу

1. Тема проекта (работы)

Разработка приложения для передачи данных в условиях нестабильного сетевого соединения с учетом их приоритетов

(утверждена распоряжением по факультету от № )

2. Срок сдачи студентом оконченного проекта (работы)

3. Исходные данные к проекту (работе)

3.1. Документация iOS. https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html

3.2. Документация по ZeroMQ: http://zguide.zeromq.org/page:all

3.3. Objective-C 2.0 и программирование для Mac («Learn Objective-C on the Mac»).Авторы: Марк Далримпл, Скотт Кнастер. Издательство: Вильямс. ISBN 978-5-8459-1607-5; 2010 г.

3.3. Набор стандартов IEEE1451

3.4. Набор стандартов JSON: ECMA-404 The JSON Data Interchange Standard.

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

4.1. Принципы приоритезации данных в системах массового обслуживания

4.2. Разработка архитектуры и функциональных алгоритмов приложения

4.3. Разработка тестовых сценариев приоритезации и передачи данных

4.4. Программная реализация и тестирование разработанного приложения

4.5. Анализ результатов

**Реферат**

ПЕРЕДАЧА, ПРИОРИТЕТ, АКТУАЛЬНОСТЬ ДАННЫХ, НЕСТАБИЛЬНОЕ СОЕДИНЕНИЕ, ПРИОРИТЕЗАЦИЯ, СМО, ОЧЕРЕДЬ СООБЩЕНИЙ, IOS, МОБИЛЬНОЕ ПРИЛОЖЕНИЕ, XCODE, IPHONE, ТРАФФИК, ZEROMQ

В этой работе рассматривается разработка приложения под ОС iOS для получения и обработки данных из другого приложения и передачи их на сервер. Предполагаются два сценария работы приложения: в условиях стабильного сетевого соединения, когда данные передаются на сервер в порядке их поступления, и в условиях нестабильного сетевого соединения, когда приложение определяет приоритеты для всех поступивших данных и отправляет в первую очередь самые важные из них.

Для создания приложения использовалась среда разработки XCode, система контроля версий Git (Source Tree). Были рассмотрены и частично применены возможные решения поставленных задач, такие как СМО, очередь сообщений, выбор приоритета инцидента на основе параметров. Использовались следующие аппаратные средства: iPhone 6 Plus, MacBook Pro.

Результатом работы является программный продукт, использующийся в существующем проекте IT компании. Проект предназначен для использования американскими полицейскими. Разработанный программный продукт представляет собой мобильное iOS приложение, отправляющее диспетчеру в первую очередь самые актуальные (а затем уже остальные) события, полученные с датчиков полицейского.

**Содержание**

Список иллюстраций 8

Список таблиц 9

Используемые определения и сокращения 10

Введение 12

Актуальность 13

Постановка цели и определение задач 15

Обзор предметной области и возможных решений 16

Экономия траффика 16

Очередь сообщений и MQ библиотек 17

Приоритетные системы массового обслуживания 25

Определение приоритета события 28

Разработка мобильного приложения 30

Архитектура проекта 30

Программные и аппаратные средства разработки 31

Формирование требований к приложению 36

Описание основных алгоритмов 36

Обработчик новых событий 39

Определение приоритета нового события 41

Выборка данных из системы для передачи на сервер 43

Взаимодействие с другим приложением 45

Интерфейс приложения 46

Передача данных с использованием ZeroMQ 47

Разработка тестовых сценариев и проведение тестирования 48

Заключение 50

Список литературы 51

Приложение 1. Программный код реализации класса ZeroMQ для передачи данных на сервер. 52

Список иллюстраций

[Рисунок 1. Общая схема очереди сообщений. 16](#_Toc421565187)

[Рисунок 2. Схема распределения задач по времени в синхронных системах. 16](#_Toc421565188)

[Рисунок 3. Схема распределения задач по времени в многопоточных синхронных системах. 17](#_Toc421565189)

[Рисунок 4. Схема распределения задач по времени в асинхронных системах. 17](#_Toc421565190)

[Рисунок 5. Общая схема системы массового обслуживания. 23](#_Toc421565191)

[Рисунок 6. Граф состояний полицейского. 24](file:///C:\Users\Maxim\Downloads\Usmanov_bachelor_v8.docx#_Toc421565192)

[Рисунок 7. Общая архитектура проекта. 25](file:///C:\Users\Maxim\Downloads\Usmanov_bachelor_v8.docx#_Toc421565193)

[Рисунок 8. Снимок экрана среды разработки XCode 6.3 26](#_Toc421565194)

[Рисунок 9. Apple iPhone 6 Plus 28](#_Toc421565195)

[Рисунок 10. Портал данных инцидентов, снимок экрана. 29](#_Toc421565196)

[Рисунок 11. Приложение для взаимодействия с датчиками, снимок экрана. 30](#_Toc421565197)

[Рисунок 12. Общая схема разрабатываемого приложения. 32](#_Toc421565198)

[Рисунок 13. Обработчик новых событий. 34](#_Toc421565199)

[Рисунок 14. Схема алгоритма выбора данных для передачи на сервер. 38](#_Toc421565200)

[Рисунок 15. Диаграмма обмена сообщениями мобильных приложений на одном устройстве. 40](#_Toc421565201)

[Рисунок 16. Пользовательский интерфейс приложения. 41](#_Toc421565202)

Список таблиц

[Таблица 1 Сравнение основных MQ библиотек. 18](#_Toc421565230)

[Таблица 2 Типы событий с приоритетами. 36](#_Toc421565231)

[Таблица 3 Состояния и их приоритеты. 37](#_Toc421565232)

[Таблица 4 Результаты тестирования приложения. 47](#_Toc421565233)

Используемые определения и сокращения

**API** (Application Programming Interface) – набор готовых классов, процедур, функций, структур и констант, предоставляемых приложением (библиотекой, сервисом) для использования во внешних программных продуктах.

**ZeroMQ** (MQ – message queue – очередь сообщений) – библиотека для обмена сообщениями.

**iOS** (iPhone Operating System) – мобильная операционная система, разрабатываемая компанией Apple для своих мобильных устройств.

**Code review** (проверка кода) – один из способов контроля качества разработки программного продукта.

**ПО** – программное обеспечение.

**ОС** – операционная система.

**IP** – интернет протокол.

**LTE** (Long-Term Evolution) - стандарт беспроводной высокоскоростной передачи данных для мобильных телефонов и других терминалов, работающих с данными.

**IEEE** (Institute of Electrical and Electronics Engineers) - Институт инженеров по электротехнике и электронике.

**POSIX** (portable operating system interface for Unix) - набор стандартов, описывающих интерфейсы между операционной системой и прикладной программой (системный API), библиотеку языка C и набор приложений и их интерфейсов.

**ОЗУ** - оперативное запоминающее устройство.

Введение

На протяжении всей своей жизни люди социально взаимодействуют между собой, обмениваются информацией. И, чем чаще происходит общение, тем больше неважной информации передается между людьми. Однако, если ограничить на некоторое время возможность связываться друг с другом, то при малейшей возможности сообщить что-либо, люди постараются передать самую важную и самую актуальную информацию.

Подобная ситуация вполне допустима у компьютерных программ при обмене данными. С развитием информационных технологий люди всё чаще стали использовать мобильные средства связи. В области сетей и телекоммуникаций человечество значительно преуспело за последнее столетие. Всё больше и больше мобильных приложений требуют доступа в глобальную сеть «Интернет» для обмена данными с другими устройствами, и современные устройства с некоторой мерой надежности могут вполне удовлетворить это требование.

Однако, в современном мире по разным причинам любая связь может быть разорвана, соединения прервано. Это может возникать из-за экранирования источников и приемников связи, удаления передатчика на большие расстояние, отключение мобильного устройства в результате разрядки аккумулятора, повреждения важных элементов, и других обстоятельств.

В данной работе будет учитываться ситуация, когда сетевое соединения нестабильно, то есть прерывается на некоторое время, но потом обязательно восстанавливается.

В проекте взаимодействуют два приложения: клиент и сервер. Задача клиента – своевременно передавать данные серверу. На выполнение этой задачи в условиях нестабильного сетевого соединения и нацелена данная работа.

Актуальность

Владение своевременной необходимой информацией - важная составляющая успешного управления любой системой. И, если такая информация содержит в себе данные об изменении состояний структурных элементов системы и должна быть передана управляющему элементу, то для такой системы актуальна данная работа.

Достаточно рассмотреть любую компанию, в которой осуществляется удаленный контроль с помощью диспетчера.

В проекте компании, занимающейся созданием оборудования для правоохранительных органов, предполагается, что у каждого полицейского имеется набор датчиков, которые считывают его состояние и отправляют данные диспетчеру в виде изменяющихся событий. Примером события может быть перемещение полицейского на 1 метр, а может быть и выстрел из пистолета. В случае, когда связь между передающим устройством мобильного сотрудника и диспетчером установлена и не прерывается, могут быть переданы данные обо всех событиях в момент, когда они произошли. Однако в этой работе рассматривается ситуация, когда по каким-либо причинам связь временно отсутствует. Тогда, после ее восстановления, необходимо передать в первую очередь самые важные и актуальные данные, так как есть угроза повторного обрыва соединения. Например, если сигнал от полицейского пропал на 2 часа после события о перемещении, то при появлении сигнала диспетчера в первую очередь будет интересовать не весь маршрут следования мобильного сотрудника, а его последнее местонахождение и последние события, обозначающие угрозу его жизни и здоровью, если таковые происходили.

Современный уровень развития сетевых технологий может обеспечить высокую скорость и стабильность передачи данных с мобильных устройств через IP соединение, однако не даёт 100% гарантий. Связь может быть прервана по причине экранирования (передатчик находится в туннеле метро и под мостом), удаленности от приёмника сигнала (за городом вдали от LTE вышки), временной неисправности (особенности работы ОС или составляющих телефона), разрядки телефона, и тому подобное.

Таким образом, данная работа актуальная для следующих систем:

* Полиция и другие государственные службы (пожарные, машины скорой помощи).
* Коммерческие организации, в которых контроль осуществляется с помощью диспетчера (такси, доставка пиццы, грузоперевозки).
* В быту (родительский контроль, отслеживание состояния больных или пожилых людей) .

Постановка цели и определение задач

Целью данной работы является разработка мобильного приложения, которое принимает данные из другого приложения, обрабатывает их, и передает на сервер в первую очередь самые актуальных из них в случае нестабильного сетевого соединения, либо все данные в случае стабильного соединения. Иными словами, после обрыва и восстановления связи приложение должно передать серверу наиболее актуальные данные. Необходимо учесть, что в данной работе актуальным обозначается событие, которое произошло позже событий того же типа.

Таким образом, в данной работе решаются следующие задачи:

* Исследовать предметную область для анализа возможных решений;
* Сформулировать требования к приложению;
* Спроектировать архитектуру приложения и разработать основные алгоритмы;
* Выбрать средства реализации и реализовать приложение для существующего проекта;
* Составление тестовых сценариев и проведение тестирования основной функциональности в рамках существующего проекта;

Обзор предметной области и возможных решений

Нестабильное сетевое соединение подразумевает разрыв связи на некоторое время, а затем обязательное ее восстановление, но на неопределенный срок. Таким образом, задача передачи данных в этом случае сводится к задаче передачи такого количества информации в промежуток времени, в который ее невозможно передать полностью.

Логично в первую очередь рассмотреть варианты экономии траффика.

Экономия траффика

Экономия трафика — комплекс мероприятий, позволяющих сократить расход интернет-трафика без ущерба для необходимого качества получаемой информации.

Рассмотрим и проанализируем общеизвестные способы экономии траффика:

* Сжатие данных. Данные обрабатываются алгоритмами сжатия, передаются в сжатом виде, а затем «распаковываются» по обратному алгоритму. Для разрабатываемого приложения такой вариант не подходит, так как сервер не способен распаковывать данные.
* Кэширование данных. Одинаковые данные не передаются. Такой вариант опять же требует доработки серверной части.
* Уменьшение качества медиа-контента. Разрабатываемое приложение не передает медиа файлы, а только текстовую информацию.
* Сокращение количества данных. В нашем случае данные должны быть переданы все. Однако, сразу после обрыва соединения разумно передавать только часть данных, самых актуальных на момент передачи.

Таким образом, варианты экономии траффика не могут решить все поставленные задачи, так как основная задача – передавать самые актуальные данные после обрыва и восстановления соединения. Требуется решение, способное создать очередь из неотправленных событий.

Архитектура проекта предлагает использовать библиотеку ZeroMQ, которая, по большому счету, выстраивает очереди сообщений и обеспечивает программисту к ней гибкий доступ.

Очередь сообщений и MQ библиотеки

Прежде чем начать разбираться с ZeroMQ, необходимо рассмотреть общую концепцию очереди сообщений.

Очередь сообщений, или, технически, FIFO(First In First Out), является одной из основных и хорошо изученных структур данных. Существуют различные реализации очереди, такие как очередь приоритетов или двусторонняя очередь, которые имеют различные свойства, но общая идея в том, что данные добавляются в очередь когда они появляются или вызывающий абонент будет готов.

Вместе с тем очередь сообщений предоставляет гарантии, что сообщение будет доставлено независимо от того, что происходит. Очередь сообщений позволяет асинхронно взаимодействовать между слабо связанными компонентами, а также обеспечивает строгую последовательность очереди. В случае недостатка ресурсов, что мешает вам сразу же обработать посылаемые данные, вы можете поставить их в очередь сообщений на сервере, который будет хранить данные до тех пор, пока клиент не будет готов.

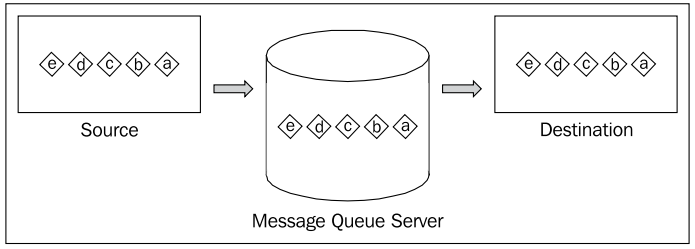


Рисунок 1. Общая схема очереди сообщений.

Очередь сообщений играет важную роль при масштабировании распределенных систем, так как поддерживает асинхронные связи. Дадим краткую информацию о разнице между синхронными и асинхронными системами.

В обычных синхронных системах задачи обрабатываются по одной за раз. Задача считается не обработанной, пока процесс ее обработки не закончен. Это самый простой способ организации работы.

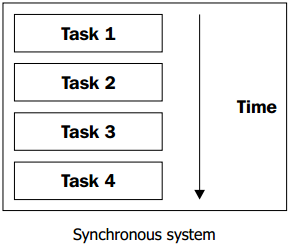


Рисунок 2. Схема распределения задач по времени в синхронных системах.

Можно также внедрить в эту систему потоки. В этом случае, процесс обработки каждой задачи выполнялся бы параллельно.

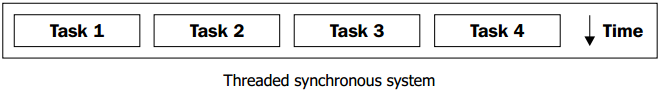


Рисунок 3. Схема распределения задач по времени в многопоточных синхронных системах.

В много-поточной модели потоки управляются самой операционной системой на одном процессоре или нескольких процессорах/ядрах.

Асинхронный ввод/вывод (AIO) позволяет программе продолжить выполнение при обработке I/O запросов. AIO является обязательным в приложениях реального времени. С помощью AIO можно обработать несколько задач в одном потоке.

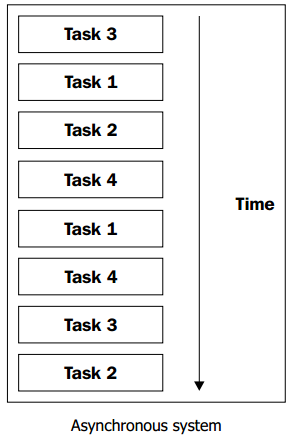


Рисунок 4. Схема распределения задач по времени в асинхронных системах.

При попытке передать самые актуальные данные, должна быть выстроена очередь с приоритетами. Удобно использовать готовые решения, представленные в виде библиотек для обмена сообщениями – MQ (meesage queue) библиотек. Их название означает «очередь сообщений», что они, собственно, и выстраивают. Рассмотрим основные MQ библиотеки.

Таблица 1 Сравнение основных MQ библиотек.

|  |  |  |  |
| --- | --- | --- | --- |
| Библиотека  Особенность | ZeroMQ | RabbitMQ | Apache ActiveMQ |
| Открытый исходный код | Lesser General Public License | Mozilla Public License | Apache License |
| Производительность | Высокая | Низкая | Средняя |
| Поддержка AMQP | - | + | + |
| Возможность задания приоритета сообщения | + | + | + |
| Возможность изменения состава очереди без потери производительности | - | - | + |
| Кроссплатформенность | + | + | + |
| Поддержка всех основных протоколов | + | - | - |
| Наличие оболочек для разных языков программирования | + | - | - |

Из таблицы 1 следует, что наиболее удобной для использования на мобильном устройстве является ZeroMQ. Однако, в ней отсутствует поддержка протокола AMQP (Advanced Message Queuing Protocol). Основная идея AMQP состоит в том, что отдельные подсистемы (или независимые приложения) могут обмениваться произвольным образом сообщениями через AMQP-брокер, который осуществляет[маршрутизацию](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%80%D1%88%D1%80%D1%83%D1%82%D0%B8%D0%B7%D0%B0%D1%86%D0%B8%D1%8F), возможно гарантирует доставку, распределение потоков данных, подписку на нужные типы сообщений. AMQP основан на трёх понятиях:

1. Сообщение (message) — единица передаваемых данных, основная его часть (содержание) никак не интерпретируется сервером, к сообщению могут быть прицеплены структурированные заголовки.
2. Точка обмена (exchange) — в неё отправляются сообщения. Точка обмена распределяет сообщения в одну или несколько [очередей](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)). При этом в точке обмена сообщения не хранятся. Точки обмена бывают трёх типов: fanout — сообщение передаётся во все прицепленные к ней очереди; direct — сообщение передаётся в очередь с именем, совпадающим с ключом маршрутизации (routing key) (ключ маршрутизации указывается при отправке сообщения); topic — нечто среднее между fanout и direct, сообщение передаётся в очереди, для которых совпадает маска на ключ маршрутизации, например, app.notification.sms.\* — в очередь будут доставлены все сообщения, отправленные с ключами, начинающимися на app.notification.sms.
3. Очередь (queue) — здесь хранятся сообщения до тех пор, пока не будут забраны клиентом. Клиент всегда забирает сообщения из одной или нескольких очередей.

Использование нескольких очередей может стать частичным решением общей задачи для передачи данных на основе их приоритета, однако не может быть полным решением, т.к. библиотеки не позволяют перестраивать очереди, если они уже сформированы, либо это происходит в ущерб производительности.

Приоритетные системы массового обслуживания

При исследовании операций часто приходится сталкиваться с системами, предназначенными для многоразового использования при решении однотипных задач. Возникающие при этом процессы получили название процессов обслуживания, а системы — систем массового обслуживания (СМО). Примерами таких систем являются телефонные системы, ремонтные мастерские, вычислительные комплексы, билетные кассы, магазины, парикмахерские и т.п.

Каждая СМО состоит из определенного числа обслуживающих единиц (приборов, устройств, пунктов, станций), которые будем называть каналами обслуживания. Каналами могут быть линии связи, рабочие точки, вычислительные машины, продавцы и др. По числу каналов СМО подразделяют на одноканальные и многоканальные.

Заявки поступают в СМО обычно не регулярно, а случайно, образуя так называемый случайный поток заявок (требований). Обслуживание заявок, вообще говоря, также продолжается какое-то случайное время. Случайный характер потока заявок и времени обслуживания приводит к тому, что СМО оказывается загруженной неравномерно: в какие-то периоды времени скапливается очень большое количество заявок (они либо становятся в очередь, либо покидают СМО необслуженными), в другие же периоды СМО работает с недогрузкой или простаивает.

Предметом теории массового обслуживания является построение математических моделей, связывающих заданные условия работы СМО (число каналов, их производительность, характер потока заявок и т.п.) с показателями эффективности СМО, описывающими ее способность справляться с потоком заявок.

В качестве показателей эффективности СМО используются: среднее число заявок, обслуживаемых в единицу времени; среднее число заявок в очереди; среднее время ожидания обслуживания; вероятность отказа в обслуживании без ожидания; вероятность того, что число заявок в очереди превысит определенное значение и т.п.

СМО делят на два основных типа (класса): СМО с отказами и СМО с ожиданием (очередью). В СМО с отказами заявка, поступившая в момент, когда все каналы заняты, получает отказ, покидает СМО и в дальнейшем процессе обслуживания не участвует (например, заявка на телефонный разговор в момент, когда все каналы заняты, получает отказ и покидает СМО необслуженной). В СМО с ожиданием заявка, пришедшая в момент, когда все каналы заняты, не уходит, а становится в очередь на обслуживание.

Для классификации СМО важное значение имеет дисциплина обслуживания, определяющая порядок выбора заявок из числа поступивших и порядок распределения их между свободными каналами. По этому признаку обслуживание заявки может быть организовано по принципу "первая пришла — первая обслужена", "последняя пришла — первая обслужена" (такой порядок может применяться, например, при извлечении для обслуживания изделий со склада, ибо последние из них оказываются часто более доступными) или обслуживание с приоритетом (когда в первую очередь обслуживаются наиболее важные заявки). Приоритет может быть как абсолютным, когда более важная заявка "вытесняет" из-под обслуживания обычную заявку (например, в случае аварийной ситуации плановые работы ремонтных бригад прерываются до ликвидации аварии), так и относительным, когда более важная заявка получает лишь "лучшее" место в очереди.



Рисунок 5. Общая схема системы массового обслуживания.

В нашем случае источником является другое приложение, а заявками – данные, поступающее от этого приложения. «Дисп1» - диспетчер постановки в очередь, «Дисп2» - диспетчер выбора заявки из очереди для подачи на прибор. Разрабатываемое приложение берет на себя задачи обоих диспетчером, очереди и прибора. Заявка считает обслуженной, если она была отправлена на сервер.

Таким образом, СМО может частично являться решением задачи с реализацией очереди сообщений. Теперь необходимо рассмотреть способы выбора приоритетов данных.

Определение приоритета события

В науке не описываются правила выбора приоритета событий, т.к. для разных систем разные события имеют разную важность. При этом существует понятие процесса «управления инцидентами», одной из задач которого является определение приоритета инцидента. Вводятся два параметра: влияние и срочность. Их произведение и определяется приоритет. Разумно для нашей системы ввести по аналогии собственные параметры. К примеру, «уровень опасности для полицейского», и другие. Для разрабатываемого приложения разумно использовать систему отслеживания состояния пользователя на основе совокупности показаний датчиков. Эта тема исследовалась одним из участников проекта, для которого реализуется приложение. Им же и был построен граф состояний, зависящих от текущего состояния и нового события (см. рис.6). Таким образом, приоритет нового события может зависеть от состояния полицейского и типа самого события.

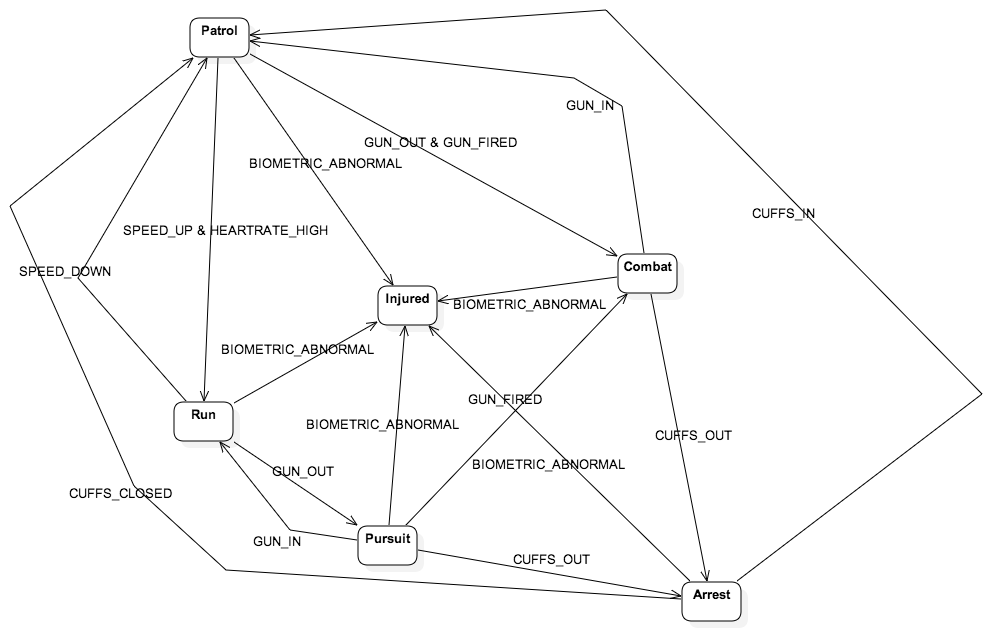


Рисунок 6. Граф состояний полицейского.

Разработка мобильного приложения

Разрабатываемое приложение является наиболее востребованным и на данный момент актуальным как составляющая часть проекта компании «Моторола Солюшнз», поэтому особенности программной реализации во многом зависят от архитектуры проекта в целом.

Архитектура проекта

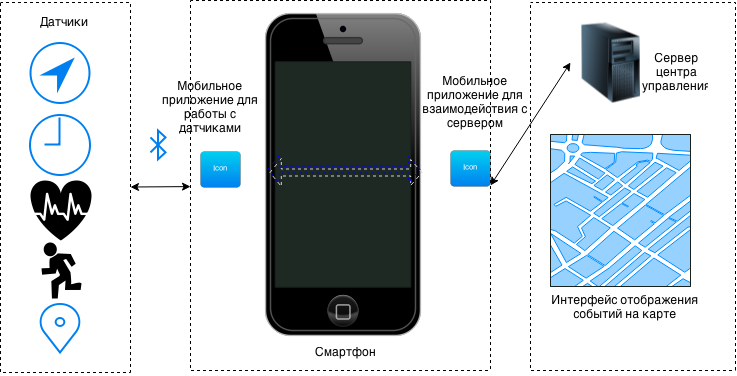


Рисунок 7. Общая архитектура проекта.

Мобильное приложение для работы с датчиками – iOS приложение, которое получает данные с различных датчиков с помощью технологии Bluetooth. Эти данные передаются в приложение, осуществляющее взаимодействие с сервером (реализуемое в данной работе), задача которого – передать эти данные на сервер (сервер центра управления) в виде событий, отображаемых через веб-интерфейс (интрерфейс отображения событий на карте).

Программные и аппаратные средства разработки

**Среда разработки XCode**

Xcode — интегрированная среда разработки программного обеспечения под OS X и iOS, разработанная корпорацией Apple.

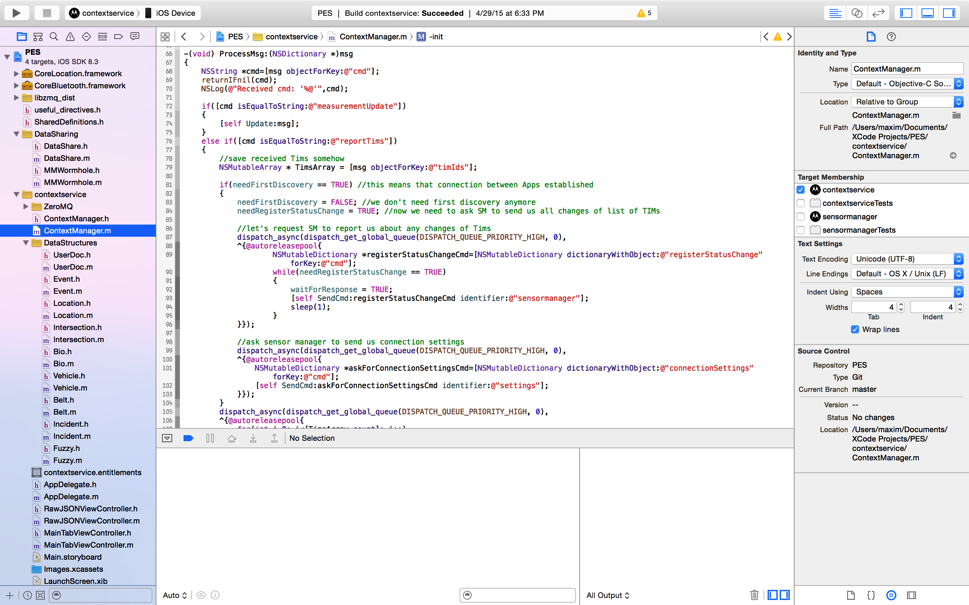


Рисунок 8. Снимок экрана среды разработки XCode 6.3

Особенности:

* распространяется бесплатно через Mac App Store.
* включает в себя большую часть документации разработчика от Apple и Interface Builder — приложение, использующееся для создания графических интерфейсов.
* включает в себя изменённую версию свободного набора компиляторов GNU Compiler Collection и поддерживает языки C, C++, Objective-C, Objective-C++ (англ.)русск., Swift, Java, AppleScript, Python и Ruby с различными моделями программирования, включая (но не ограничиваясь) Cocoa, Carbon и Java.

**Мобильная операционная система iOS**

iOS — операционная система для смартфонов, электронных планшетов и носимых проигрывателей, разрабатываемая и выпускаемая американской компанией Apple. Была выпущена первоначально — для iPhone и iPod touch, позже — для таких устройств, как iPad и Apple TV. В отличие от Windows Phone (Microsoft) и Android (Google), выпускается только для устройств, производимых фирмой Apple.

* Пользовательский интерфейс iOS основан на концепции прямого манипулирования с использованием жестов «мультитач». Элементы управления интерфейсом состоят из ползунков, переключателей и кнопок.
* iOS разработана на основе OS X и использует тот же набор основных компонентов Darwin, совместимый со стандартом POSIX.

**Смартфон iPhone 6 Plus**

iPhone 6 Plus — смартфон корпорации Apple, работающий на iOS 8.

* Сотовая связь: GSM/EDGE, UMTS/HSPA+/DC-HSDPA, CDMA/EV-DO, LTE.
* Интерфейсы: Wi-Fi (802.11 a/b/g/n/ac), Bluetooth 4.0, GPS, ГЛОНАСС; NFC (Apple Pay).
* Процессор: Apple A8/M8.
* ОЗУ: 1 ГБ LPDDR3.
* Флеш-память: 16 ГБ/64 ГБ/128 ГБ.
* Аккумулятор: Li-ion 2915 мА·ч.
* Ввод: микрофон, сенсорный экран, датчик ускорения и ориентации, компас, TouchID.



Рисунок 9. Apple iPhone 6 Plus

**Библиотека ZeroMQ**

ZeroMQ – открытая свободно распространяемая высокопроизводительная библиотека сообщений, которая помогает разработчикам создавать распределенные и параллельные приложения на высоком уровне.

В разрабатываемом приложении должна обязательно использоваться для коммуникации с сервером (из требований архитектуры проекта).

**Серверное приложение с веб-интерфесом для диспетчера**

Данные, отправленные из разрабатываемого приложения, обрабатываются серверным приложением и отображаются на карте в виде событий.

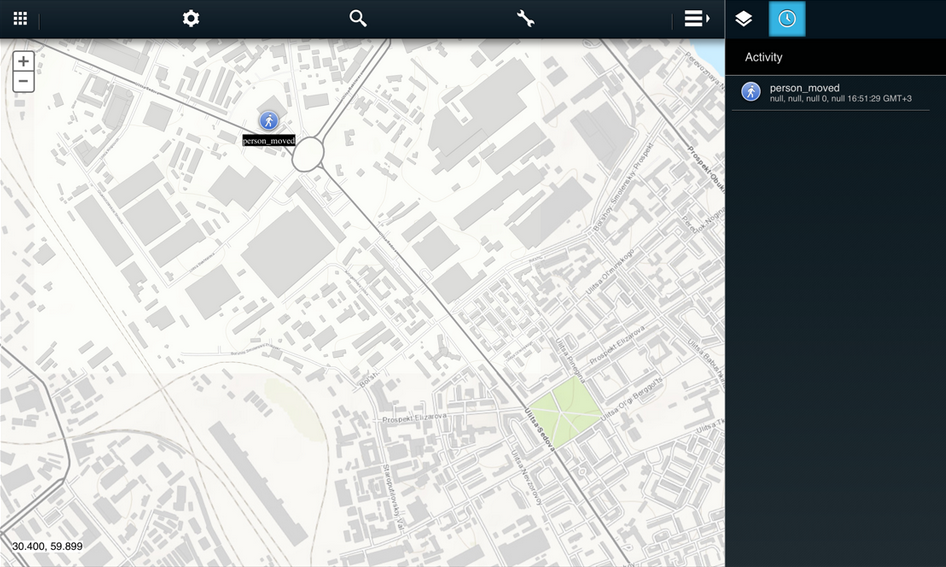


Рисунок 10. Портал данных инцидентов, снимок экрана.

**Приложение для взаимодействия с датчиками**

Приложение для iOS, которое получает информацию с датчиков и передает ее разрабатываемому приложению в обработанном виде.

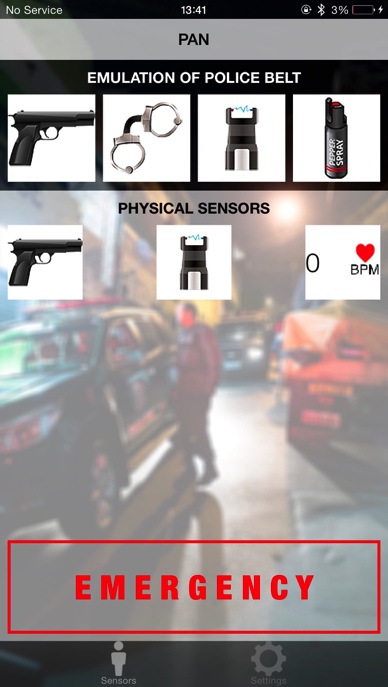


Рисунок 11. Приложение для взаимодействия с датчиками, снимок экрана.

Формирование требований к приложению

При разработке приложения необходимо учитывать то, что впоследствии оно будет интегрировано в существующий проект, в котором оно востребовано. Поэтому стоит сформировать отдельно общие и проектные требования. К общим можно отнести:

* Передача актуальных данных после возобновления соединения (независимо от времени, на которое оно было прервано);
* Сохранение всех событий. Рано или поздно все события должны быть переданы на сервер;
* Совместимость с любым сервером;
* Общий принцип определения приоритета события для ограниченного количества типов событий;
* Высокая производительность (мобильное приложение обычно не имеет больших вычислительных ресурсов);

Проектными требованиями являются:

* Взаимодействие с другим приложением (от которого поступают данные) в соответствии со стандартом IEEE 1451;
* Приложение для операционной системы iOS;
* Передача данных на сервер по протоколу ZMTP;
* Выполнение общих требований для заданного набора событий;

Описание основных алгоритмов

Рассмотрев существующие возможные решения, разумно прийти к выводу, что для реализации приложения удобно использовать измененную схему системы массового обслуживания. Отличие от традиционной СМО заключается в том, что в разрабатываемом приложении реализовано две очереди: актуальных и неактуальных событий (см. Рисунок 12). Под актуальными следует понимать события, которые появились позже остальных событий такого же типа. Соответственно, неактуальными являются остальные события.

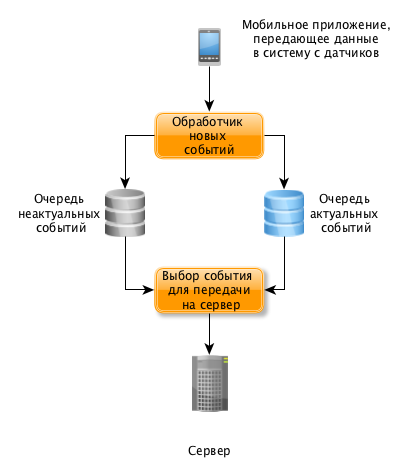


Рисунок 12. Общая схема разрабатываемого приложения.

На схеме (Рисунок 12) показана общая структура разрабатываемого приложения, которое в рамках проекта имеет название Context Service. К нему на вход в произвольные моменты времени поступают данные (события). Для определения их приоритета и постановки в соответствующую очередь реализован алгоритм обработчика новых событий, который будет представлен далее.

Необходимость использования очередей объясняется тем, что на вход приложения новые события могут поступать длительное время в большом количестве, в то время как соединение с сервером может быть разорвано или ограничено. Основная задача приложения – передать на сервер самые актуальные события. Для них используется отдельная очередь. Логично предполагать, что, если соединение было разорвано и вновь установлено, то оно может быть разорвано повторно в любой момент. Поэтому сначала передаются данные из очереди актуальных событий, затем остальные. Работу выбора данных и отправки их на сервер выполняет блок «Выбор события для передачи на сервер», алгоритм которого также будет рассмотрен подробнее.

Мобильное приложение, которое получает данные от датчиков, передает в разрабатываемое приложение эти данные, которые, в свою очередь, воспринимаются, как события, влияющие на состояние пользователя. В рамках существующего проекта для пользователя (полицейского) определены следующие состояния: нейтральное, бег, погоня, ранение, бой, арест. Они являются вершинами графа, соединенными дугами событий (Рисунок 10). Именно на основе этого графа определяются приоритеты событий. Об этом подробнее написано далее.

### Обработчик новых событий

На рисунке 13 представлен алгоритм обработчика новых событий, поступающих из приложения, взаимодействующего с датчиками. Для начала, определяется новое состояние полицейского в соответствии с графом состояний (Рисунок 6).

Поскольку любое новое событие является самым «свежим», то среди однотипных событий его можно считать самым актуальным (события и их типы представлены в таблице 1) и заменить им «устаревшее» событие того же типа.

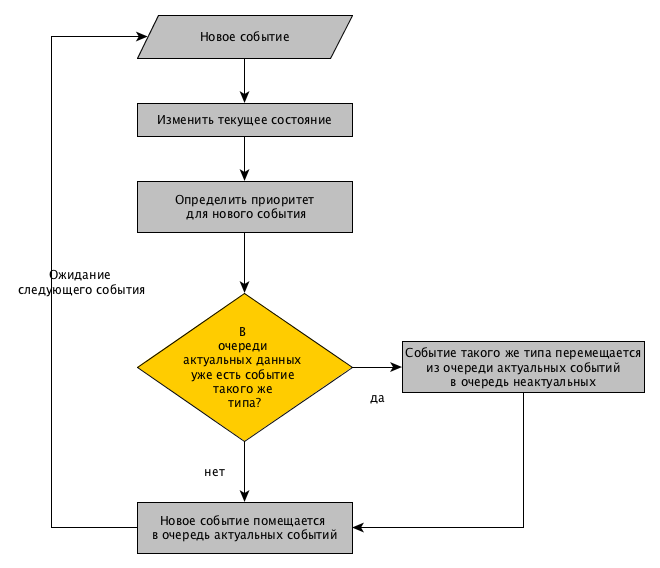


Рисунок 13. Обработчик новых событий.

Очередь неактуальных событий организована по принципу «FIFO» (First In First Out – первый пришел, первым ушел). Поэтому «устаревшее» событие встраивается в эту очередь между двумя событиями, которые произошли до и после него.

### Определение приоритета нового события

События, определяемые датчиками, могут быть по-разному важными, однако одно и то же событие может иметь различную важность при разных обстоятельствах. Например, полицейский может достать пистолет во время бега, а может достать в нейтральном состоянии. В первом случае это будет означать погоню, в другом – потенциальную перестрелку.

Таблица 2 Типы событий с приоритетами.

|  |  |  |
| --- | --- | --- |
| **Тип события** | **Приоритет типа** | **События** |
| Пистолет | 6 | достать из кобуры(4), вернуть в кобуру(2), выстрелить(5) |
| Перцовый баллончик | 5 | достать из кобуры(3), вернуть в кобуру(1) |
| Шокер | 5 | достать из кобуры(3), вернуть в кобуру(1) |
| Наручники | 4 | достать из кобуры(2), вернуть в кобуру(1) |
| Экстренный вызов | 7 | экстренный вызов(7) |
| Биометрический | 5 | нормальное(1), ненормальное состояние(3) |
| Пульс | 3 | низкий(2), высокий(3), средний(1) |
| Перемещение | 1 | перемещение(0), повышение(2) и понижение скорости(1) |

По аналогии с определением приоритета инцидента, приоритет события также будет высчитываться через параметры. В качестве параметров удобно использовать приоритет состояния (таблица 3), приоритет типа события (таблица 2) и приоритет самого события (таблица 2, в скобках). Тогда итоговым значением будет выступать их сумма.

Таблица 3 Состояния и их приоритеты.

|  |  |
| --- | --- |
| **Состояние** | **Приоритет состояния** |
| Нейтральное | 1 |
| Бег | 2 |
| Погоня | 3 |
| Ранение | 6 |
| Бой | 6 |
| Арест | 4 |

### Выборка данных из системы для передачи на сервер

В связи с особенностями архитектуры проекта и операционной системы iOS, узнать о том, что соединение с сервером установлено, возможно только попытавшись отправить данные. Поэтому приложение постоянно пытается отправить данные на сервер (если они есть) в отдельном потоке.

Чтобы передать сначала самые актуальные данные, данные выбираются из очереди актуальных событий. При этом выбрано будет событие с самым высоким приоритетом. (Рисунок 14).

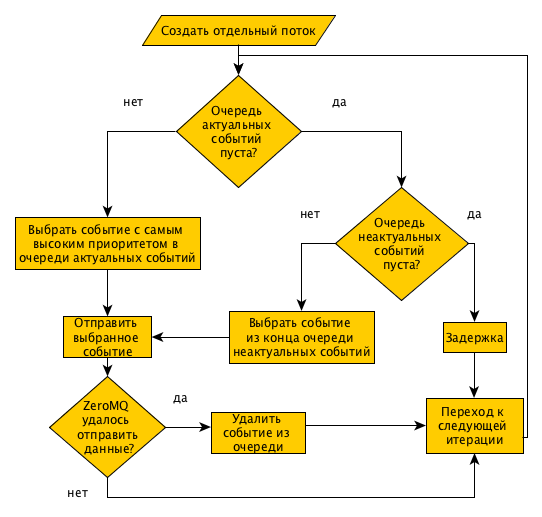


Рисунок 14. Схема алгоритма выбора данных для передачи на сервер.

Если первая очередь пуста, то события для отправки будут браться из конца очереди неактуальных событий (по принципу FIFO).

Успешное переданное событие будет удалено из очереди, из которой оно было выбрано. В противном случае – оно там остается, а приложение снова пытается отправить на сервер данные, актуальные на момент следующей итерации.

Однако, возможны и такие случаи, когда обе очереди оказываются пустыми. Тогда, при переходе к следующей итерации, устанавливается задержка, чтобы процессор не использовался впустую и не расходовал много электроэнергии.

### Взаимодействие с другим приложением

IEEE 1451 — набор стандартов для взаимодействия с «умными» датчиками, разработанный институтом инженеров по электротехнике и электронике (IEEE). В нём описывается не только набор открытых, независимых от сетей интерфейсов для подключения датчиков, но и алгоритмы взаимодействия встраиваемых модулей с основной программой. В данном проекте обмен данными между Context Service и Sensor Manager должен осуществляться в соответствии с этим стандартом (Рисунок 15).

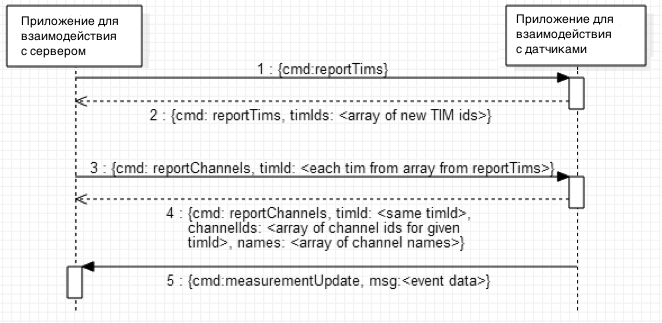


Рисунок 15. Диаграмма обмена сообщениями мобильных приложений на одном устройстве.

Инициатором взаимодействия выступает приложение, взаимодействующее с сервером (ПВС). Оно запрашивает у приложения, взаимодействующего с датчиками (ПВД), идентификаторы подключенных датчиков. В ответном сообщении он получает массив этих идентификаторов, для каждого из которых запрашивает имена каналов – конкретные назначения датчиков. На основе полученной информации ПВС способно определить тип события, которое приходит от другого приложения лишь в сопровождении идентификатора датчика и номера канала.

Интерфейс приложения

Пользовательский интерфейс приложение представляет собой три организованные вкладки (Рисунок 16): настройки соединения с сервером (IP адрес и порты сервера), лог данных, передающихся между приложениями, и лог данных, передаваемых на сервер.

Изначально архитектура проекта предполагала реализацию Context Service в виде фонового процесса, не имеющего графического интерфейса. Однако, в iOS всё приложения имеют графическую оболочку, поэтому в данном приложении интерфейс используется только для вывода отладочной информации.

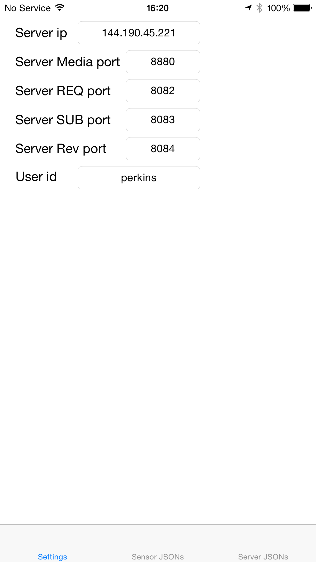


Рисунок 16. Пользовательский интерфейс приложения.

Передача данных с использованием ZeroMQ

Для передачи данных на сервер была выбрана библиотека ZeroMQ. Она является удобной оболочкой для работы с сокетами. Она избавляет программиста от необходимости опускаться на низкий уровень, позволяя уделять больше внимания функциональной логике приложения.

Первое, что стоит узнать о ZeroMQ, это то, что она не является традиционной системой очередей сообщений, так как дает инструменты для создания собственной системы очередей сообщений через библиотечный набор вызовов.

В архитектуре проекта ZeroMQ выбрана для коммуникации приложений не случайно. Библиотека имеет ряд преимуществ.

* работает на различных архитектурах от ARM до Itanium и поддерживается более чем в 20 языках программирования.
* позволяет делать некоторые асинхронные операции ввода/вывода, также ZeroMQ может ставить в очередь сообщений сообщения из потока ввода/вывода. Потоки ввода/вывода в ZeroMQ работают асинхронно при общении с сетевым трафиком, поэтому они могут делать и остальную работу. ZeroMQ позволяет гораздо упростить работу с сокетами.
* Выокая производительность. Веб-сайту Second Life удалось получить 13,4 микросекунды непрерывного времени ожидания и до 4 100 000 сообщений в секунду. ZeroMQ может использовать многоадресный транспортный протокол, который является эффективным методом для передачи данных в различных направлениях.
* Библиотечный набор функций обеспечивает сокеты (обобщенные сокеты по традиционным IP и Unix domain socket), каждый из которых может представлять собой связи «многие-ко-многим» между конечными точками. При работе с сообщениями, для сокетов требуется, чтобы была использована модель сообщений (pattern), и особенно оптимизирована для такого рода паттерна.

Основные паттерны:

* Request–reply. Соединяет набор клиентов с набором услуг: удаленный вызов процедур и паттерн распределения задач.
* Publish–subscribe. Для подключения множества публикаций с множеством подписчиков. Это паттерн распределения данных.
* Push–pull (pipeline). Подключение узлов в паттерн, который может иметь несколько шагов, и циклы. Это паттерн сбора и параллельного распределения задач.
* Exclusive pair. Соединяет два сокета в эксклюзивную пару. (Это расширенный шаблон низкого уровня для конкретных случаев применения.)

Каждый паттерн определяет конкретную топологию сети. Запрос-ответ определяет так называемый "автобус", публикации-подписки определяет "распределение данных в виде дерева", двухтактный определяет "parallelised трубопровода". Все паттерны намеренно разработаны таким образом, чтобы быть абсолютно гибкими при разработке.

Любое сообщение через сокет рассматривается как непрозрачный набор данных. Доставка абоненту может быть автоматически отфильтрована ведущей строкой набора данных. Доступна передача сообщений с TCP, PGM (надежная Multicast), взаимодействие между процессами (IPC) и по межпоточной связи (ITC).

Основная библиотека ØMQ имеет хорошую производительность из-за своей внутренней модели потоков, и может превзойти обычные приложения TCP с точки зрения пропускной способности за счет использования техники автоматического дозирования сообщений.

ZeroMQ реализует ZMTP, ZeroMQ Message Transfer Protocol. ZMTP определяет правила для обратной совместимости, расширяемых механизмов безопасности, команды и оформлении сообщения, метаданные соединения и другие функциональнии на транспортном уровне.

При создании очередей ZeroMQ позволяет задавать приоритет пакетам через функцию zmq\_setsockopt (опция ZMQ\_TOS ). Однако, без изменения самой библиотеки, невозможно изменять приоритеты пакетов, уже добавленных в очередь. К примеру, последние два события будут «экстренный вызов». Разумно отправлять в первую очередь только последнее событие. Для ZeroMQ оба будут иметь одинаковый приоритет, поэтому оба будут отправлены, что для нас не желательно.

Иными словами, использование такой библиотеки позволяет сохранить принцип простоты реализуемости разрабатываемого приложения на любой платформе, независимо от используемого языка программирования.

Таким образом, ZeroMQ позволяет задавать приоритет пакету, который мы собираемся отправить. Он добавляется в очередь сообщений, которую формирует библиотека. Однако, подразумевается, что сообщение будет отправлено мгновенно, либо с незначительной задержкой, поэтому нет необходимости создавать методы, позволяющие изменять состав очереди сообщений ZeroMQ. Такой подход сильно усложняет реализацию алгоритма отправки актуальных данных. Поэтому очереди были реализованы отдельно от библиотеки ZeroMQ, и задача ZeroMQ ограничивается лишь непосредственной отправке указанных событий на сервер, и уведомлении об успешности выполнения этой процедуры.

В приложении 1 представлен код реализации класса ZeroMQ, который содержит следующие методы:

* +(void) initialize;

Функция инициализации экземпляра класса.

* +(BOOL) sendNSStringAsJSON:(NSString\*)data toServer:(NSString\*)serverIpAddr port:(NSInteger)port;

Передает JSON данные data, предоставленные в виде строки на сервер с IP serverIpAddr и портом port. Возвращает TRUE в случае успешной передачи, и FALSE – в обратном случае.

* +(BOOL) sendNSDataAsJSON:(NSData\*)data toServer:(NSString\*)serverIpAddr port:(NSInteger)port;

Передает JSON данные data, предоставленные в виде NSData на сервер с IP serverIpAddr и портом port. Возвращает TRUE в случае успешной передачи, и FALSE – в обратном случае.

* +(BOOL) sendNSDictionaryAsJSON:(NSDictionary\*)dict toServer:(NSString\*)serverIpAddr port:(NSInteger)port;

Передает JSON данные data, предоставленные в виде словаря (типа ключ:значение) на сервер с IP serverIpAddr и портом port. Возвращает TRUE в случае успешной передачи, и FALSE – в обратном случае.

Для передачи данных необходимо сначала открыть соединения. Сначала открывается сокет с помощью функции zmq\_socket. Через него устанавливается соединение с сервером по указанному адресу и порту, для чего используется функция zmq\_connect.

После этого можно передавать данные. Для этого вызывается функция zmq\_send. Об успешности передачи можно судить только по ответу от сервера, который получается с помощью функции zmq\_recv.

По завершению процедуры необходимо разорвать соединение (zmq\_disconnect) и закрыть используемый сокет (zmq\_close).

Разработка тестовых сценариев и проведение тестирования

Для контроля качества разработанного приложения использовались code review (проверка кода другими участниками проекта) и частичное полуавтоматическое тестирование.

Каждый тестовый сценарий подразумевает наличие входного набора данных, обеспечивающее максимальный охват программного кода, и соответствующего набора предполагаемых выходных данных.

Таким образом были частично протестированы следующие функциональные особенности (таблица 4):

Таблица 4 Результаты тестирования приложения.

|  |  |  |
| --- | --- | --- |
| **Номер теста** | **Описание** | **Результат** |
| 1 | Обработчик новых событий | Пройден |
| 2 | Выбор приоритета для события на основе состояния полицейского | Пройден |
|  | Выбор данных из очереди для отправки: | Пройден |
| 4 | - только очередь активных событий пуста | Пройден |
| 5 | - обе очереди пустые | Пройден |
| 6 | - при постоянном поступлении в систему новых событий | Пройден |
| 7 | Работа системы при обрыве и восстановлении соединения с сервером | Пройден |

Заключение

В ходе работы было разработано мобильное iOS приложение, которое удовлетворяет всем установленным требованиям:

* взаимодействует с других приложением в соответствии с API, указанным в стандарте IEEE 1451 и получает сообщения о новых событиях от этого приложения;
* определяет приоритета события, основываясь на его типе и на текущем состоянии полицейского;
* выстраивает упорядоченные очереди актуальных и неактуальных событий;
* обеспечивает передачу самых актуальных данных на сервер в случае наличия соединения с ним (не зависимо от продолжительности отсутствия соединения перед этим);
* обеспечивает сохранность всех событий (абсолютно все данные будут переданы, но в порядке очереди).

Кроме того, приложение имеет графический интерфейс для отображения отладочной информации. Также он позволяет устанавливать настройки для соединения с сервером.

Для контроля качества приложения была проведена частичная проверка кода другими участниками проекта и частичное тестирование основных функций приложения.

Таким образом, все поставленные задачи в ходе работы были выполнены, а цель работы можно считать достигнутой.

Список литературы

1. Милославская Н.Г., Сенаторов М.Ю., Толстой А.И., Управление инцидентами информационной безопасности и непрерывностью бизнеса, 2-е издание, исправленное, 2014 – 170 с.
2. Документация ZeroMQ – URL:

<http://zguide.zeromq.org/>. (дата последнего обращения 02.05.2015)

1. Документация iOS – URL: <https://developer.apple.com/library/ios/documentation/iPhone/Conceptual/iPhoneOSProgrammingGuide/Introduction/Introduction.html>. (дата последнего обращения 25.05.2015)
2. Pieter Hintjens. ZeroMQ. Messaging for Many Applications, 2013 – 516 с.
3. Набор стандартов JSON: ECMA-404 The JSON Data Interchange Standard.
4. Faruk Akgul. ZeroMQ.Use ZeroMQ and learn how to apply different message patterns., Packt Publishing, 2013 – 108 с.
5. Бочаров П.П., Печинкин А.В.. Теория массового обслуживания. — М.: РУДН, 1995. — С. 530.
6. Матвеев В. Ф., Ушаков В. Г. Системы массового обслуживания, 1984 – 240.
7. Набор стандартов IEEE1451.
8. Objective-C 2.0 и программирование для Mac («Learn Objective-C on the Mac»).Авторы: Марк Далримпл, Скотт Кнастер. Издательство: Вильямс. ISBN 978-5-8459-1607-5; 2010 г.

Приложение 1. Программный код реализации класса ZeroMQ для передачи данных на сервер.

// ZeroMQ.m

// PES

//

// Created by Maxim on 2/13/15.

// Copyright (c) 2015 motorolasolutions. All rights reserved.

//

#import <Foundation/Foundation.h>

#import "DataShare.h"

#import "ZeroMQ.h"

#import "zmq.h"

@interface ZeroMQ ()

@end

@implementation ZeroMQ

void \* zmqContext;

+(void) initialize

{

[super initialize];

zmqContext = zmq\_ctx\_new();

}

+(BOOL) sendNSStringAsJSON:(NSString\*)data toServer:(NSString\*)serverIpAddr port:(NSInteger)port

{

//prepare connection string using server ip and port

NSString \* connectionURL = [NSString stringWithFormat:@"tcp://%@:%ld", serverIpAddr, (long)port];

//open ZeroMQ connection

void \* zmqRequester = zmq\_socket (zmqContext, ZMQ\_REQ);

int connectionResult = zmq\_connect (zmqRequester, [connectionURL UTF8String]);

NSLog(@"[ZMQ] Connect returned status: %d", connectionResult);

//abort send if no connection

if(connectionResult != 0)

{

NSLog(@"[ZMQ] Connection failed. Check your internet connection and if server ip and port are correct.");

//close ZeroMQ connection

zmq\_disconnect (zmqRequester, [connectionURL UTF8String]);

zmq\_close (zmqRequester);

return FALSE;

}

//send data and receive response

NSLog(@"[ZMQ] Sending Data…\n");

char responseBuffer[ZMQ\_RESPONSE\_BUFFER\_SIZE];

zmq\_send (zmqRequester, [data UTF8String], [data length], 0);

zmq\_recv (zmqRequester, responseBuffer, ZMQ\_RESPONSE\_BUFFER\_SIZE, 0);

NSLog(@"[ZMQ] Received '%c%c%c' OK\n",responseBuffer[0],responseBuffer[1],responseBuffer[2]);

//close ZeroMQ connection

zmq\_disconnect (zmqRequester, [connectionURL UTF8String]);

zmq\_close (zmqRequester);

//check response

if(responseBuffer[0]== 'A' && responseBuffer[1] == 'C' && responseBuffer[2] == 'K')

return TRUE;

else return FALSE;

}

+(BOOL) sendNSDictionaryAsJSON:(NSDictionary\*)dict toServer:(NSString\*)serverIpAddr port:(NSInteger)port

{

//dbg: NSLog(@"[ZMQ] Sending NSDictionary as JSON");

//convert NSDictionary to JSON string

NSData \* data2send = [DataShare JSON: dict];

//send as NSString

return [self sendNSDataAsJSON:data2send toServer:serverIpAddr port:port];

}

+(BOOL) sendNSDataAsJSON:(NSData\*)data toServer:(NSString\*)serverIpAddr port:(NSInteger)port

{

//dbg: NSLog(@"[ZMQ] Sending NSDictionary as JSON");

//convert NSData to JSON string

NSString \* data2send = [DataShare rawJSONString:data];

//send as NSString

return [self sendNSStringAsJSON:data2send toServer:serverIpAddr port:port];

}

@end