Санкт-Петербургский Государственный университет

Математико-механический факультет

Кафедра системного программирования

Разработка протокола прикладного уровня для управления системами

Курсовая работа студента 344 группы

Логиновой Виты Владимировны

Научный руководитель Абусалимов Э. Ш.

аспирант кафедры системного программирования

Санкт-Петербург

2013

Оглавление

Введение

Управляемые системы - это системы, которые под воздействием некоторого управления могут переходить из одного состояния в другое. Понятие управляемой системы включает в себя широкий класс объектов, таких как живые организмы, механические и электронные устройства и многое другое. Нас интересуют электронные устройства с обратной связью. Важной частью всех управляемых систем является именно управление, то есть то, как управляющее устройство будет передавать сообщения системе.

На данный момент имеется большое число электронных устройств разного типа, которые поддерживают разные протоколы передачи, приспособленные для конкретного случая. Впервые с этой проблемой я столкнулась при разработке мобильного приложения под Windows Phone 7, которое управляет роботом Lego NXT и DIY-машинкой по Wi-Fi. Одно из требований к приложению - легкая расширяемость под другие протоколы. Выполнить это требование - задача нетривиальная, ведь протокол управления для каждого протокола обладает особой спецификой. Обилие таких ad hoc протоколов сильно усложняет разработку программы для управляющих устройств. По факту, управляющему устройству приходится поддерживать сразу несколько приложений - по одному на каждую систему. В связи с чем возникла идея каким-то образом обобщить управляющие протоколы.

Постановка задачи

Существующий на момент выполнения данной работы протокол, используемый в проекте Embox для управления роботами - временное решение. Он был простым, но ограниченным, плохо расширяющимся. Коме того, разные устройства требовали поддержку разных протоколов, что значительно усложняло процесс разработки управляющей программы. Целью работы является разработка унифицированного протокола передачи данных для устройств с дистанционным управлением, который будет лишен вышеперечисленных недостатков.

Кроме того, протокол должен удовлетворять следующим требованиям:

- Протокол должен быть простым и легко расширяться. То есть реализация поддержки каждого нового устройства, имеющего свой уникальный протокол, должна быть несложной
- Клиент должен уметь подстраиваться под возможности устройства
- Отсутствие избыточной информации. Избыточная информация не только занимает лишнюю память, но так же тратит дополнительное время на передачу данных, что критично для систем реального времени

Обзор существующих решений

В настоящее время существуют решения, которые позволяют на этапе соединения управлять параметрами сессии, а также изменять настройки уже непосредственно после установления соединения. В ходе поиска аналогичных решений были рассмотрены USB[1] и SIP[2].

USB

В передачи данных с помощью интерфейса USB участвуют две стороны с разными ролями: хост (ведущий) и устройство (ведомый). Когда устройство USB подключается к хосту, начинается так называемый enumeration proccess, с помощью которого хост определяет устройство и выясняет необходимый драйвер. Процесс проходит в несколько этапов:

- хост отправляет повторяющиеся сигналы подключенному устройству. В процессе этого также выясняется скорость передачи
- хост считывает информацию с устройства
- устройство получает адрес, состоящий из семи бит

Все передачи проходят через виртуальные каналы, соединяющие конечные точки устройства с хостом. После установления соединения каждая точка возвращает дескриптор - структуру данных, которая содержит конфигурацию и ожидания.

SIP

SIP - протокол, который используется для создания, управления и прерывания сессии в сетях, основанных на IP. Его основные задачи:

- SIP может установить местоположение пользователя
- Участникам сессии могут договариваться об использовании каких-либо возможностей, а также менять их во время самой сессии
- Управления конференцией

SIP поддерживает 14 команд, основных из которых всего 6. Интересна команда OPTIONS, которая запрашивает информацию о функциональности сервера. Пример сценария установления соединения при помощи SIP показан на рис. 1.

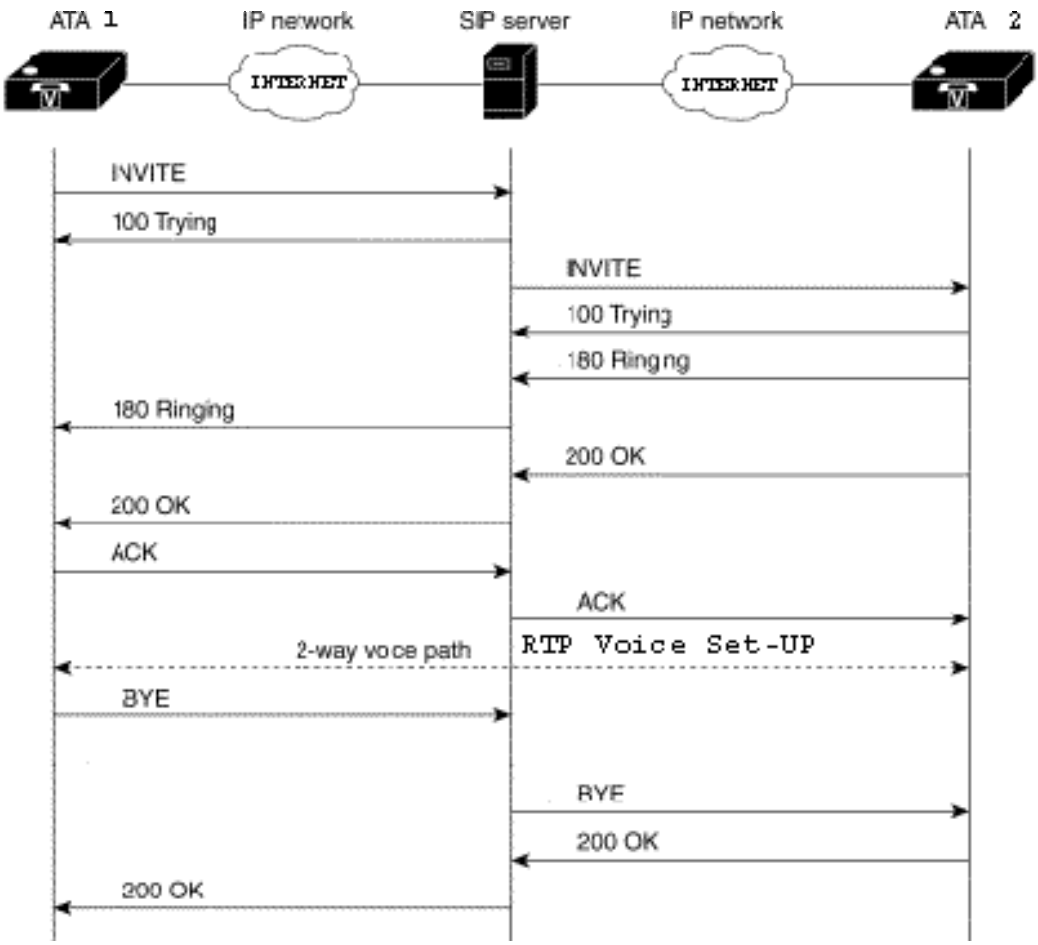

Рис.1 Пример сценария установления соединения в SIP

Результат обзора

Оба рассмотренных решения обладают способностью договариваться об оптимальном способе взаимодействия между участниками сессии. Кроме того, SIP позволяет менять конфигурацию во время сессии. Использовать в чистом виде данные решения, конечно, нельзя: недостаток USB в приложении к рассматриваемой проблеме - специфичность (низкий уровень решаемых задач - управление питанием, коммутацией). У SIP же есть другой важный недостаток: избыточность информации, обусловленная использованием формата HTTP. Тем не менее, решено было позаимствовать вышеупомянутые концепции.

Описание решения

В ходе выполнения работы был разработан протокол прикладного уровня. Его главная и единственная задача - определить способ взаимодействия между клиентом и устройством, а также само взаимодействие. Протокол не решает таких проблем, как гарантия доставки сообщений, поскольку данную задачу уже выполняют протоколы более низкого уровня (TCP при передаче по Wi-Fi, RFCOMM при передаче по Bluetooth). В предложенном решении используется протокол, условно разделенный на две части: на часть, управляющую соединением, и на часть, соответствующую протоколу обмена.

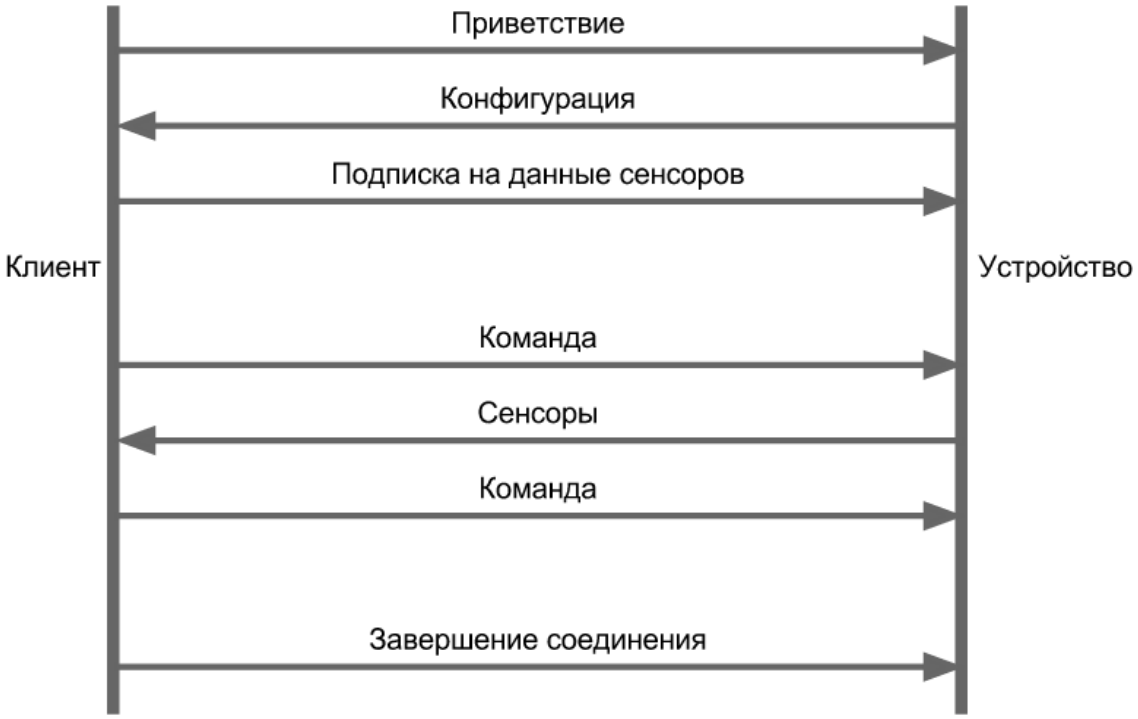

Рис. 2 Сценарий взаимодействия разработанного протокола

На высоком уровне абстракции взаимодействие клиента и устройства выглядит так, как показано на рис. 2. Сначала клиент посылает приветствие - запрос на разрешение управлением. Если устройство отказывает в доступе, то соединение закрывается. Например, устройство не дает подключиться к себе, если им кто-то уже управляет.

В ответ на приветствие устройство сообщает о своей конфигурации клиенту. Конфигурация - самая важная часть, по ней клиент понимает, какие, например, команды доступны для

управления устройством. Также конфигурация содержит информацию о типе клиента и протоколе обмена, который будет использоваться в дальнейшем.

После того, как устройство отправило свою конфигурацию, соединение считается установленным. Теперь клиент может управлять устройством путем исполнения доступных команд, а также запрашивать данные с сенсоров. В любой момент клиент может отписаться от получения данных сенсоров или же наоборот, подписаться на них. То, в каком виде будут посылаться данные с сенсоров и команды, зависит от определенного ранее протокола обмена.

Сессия заканчивается закрытием соединения.

Протокол управления соединением

Для управления соединением нам достаточно передавать структуры данных. В дипломе Александра Константинова[3] приведен обзор популярных технологий для кодирования структур. В обзоре рассмотрены: Hessian, ASN.1, Protobuf, Kryo, Protostuff и Avro. Результаты сравнения по таким параметрам, как возможность получить решение для нескольких языков и платформ, наличие у технологии сообщества, эффективное кодирование данных и простота использования и интеграции, приведены в таблице (табл. 1).

	Мультиплатформенность	**Community**	**Эффективное хранение**	**Простота**
Hessian	+	-	+	+
ASN.1	+	+	+	-
Protobuf	+	+	+	+
Kryo	-	-	+	+
Protostuff	+	-	+	+
Avro	-	-	+	+

Таблица 1. Сравнение существующих технологий

(выдержка из табл. 1 диплома Александра Константинова)

Из всех рассмотренных вариантов решено использовать именно Protocol Buffers[4], так как он обладает всеми перечисленными ниже качествами:

- Protobuf имеет широкую техническую поддержку (разработан корпорацией Google).

Данные, передаваемые по протоколу, сгененрированному посредством Protobuf, эффективно кодируются

- Protobuf имеет реализацию для 25 языков программирования, большая часть из которых, правда, предлагается сторонними разработчиками. Google предлагает решения для языков Python, Java и C++. Таким образом, практически для любого устройства можно реализовать поддержку protobuf
- Protobuf обладает подробной документацией и прост в использовании

Основной единицей взаимодействия между клиентом и устройством является сообщение. Каждое сообщение описывается в proto-файле на специальном языке и потом обрабатывается компилятором protobuf, на выходе получается код сериализатора и десериализатора на нужном языке. Таким образом, каждому типу сообщения, который используется в протоколе (рис. 2), соответствует свое описания на языке protocol buffers. Наиболее содержательным является сообщение, описывающее конфигурацию устройства, ее и рассмотрим подробнее.

```
message DeviceConfigurationMessage {
     required string id = 1;
     required enum type = 2;
     required bytes sensors = 3;
     required bytes commands = 4;
}
```

- id - поле, содержащее уникальный идентификатор устройства
- type - тип устройства
- sensors и commands - массивы байтов, которые содержат флаги, ассоциированные с сенсорами и командами из списка. Так как наборы команд и сенсоров для разных устройств могут значительно различаться, то решено было хранить списки для каждого устройства отдельно

Протокол обмена

Протокол обмена может различаться для разных устройств. Именно это позволяет поддерживать устройства с удобными для них протоколами. Для Lego NXT, например, удобно использовать формат direct-команд, используемый в оригинальной прошивке.

Иногда может быть удобно в качестве протокола обмена тот же protobuf. Например, при

прототипировании, когда необходимо протестировать основную функциональность, не нужно поддерживать дополнительные библиотеки в режиме исполнения.

Реализация и внедрение

После того, как основная идея протокола была очерчена и разработано описание, было необходимо реализовать решение на практике. Так как разработанный протокол предусматривает возможность использования разных форматов обмена данными, а поддержка возлагается на клиент, то его доработка является самой объемной и сложной частью работы. Поэтому в первую очередь было решено реализовать симулятор, с помощью которого была налажена работа обновленного клиента. Затем протокол нужно было внедрить на целевое устройство - Lego NXT под управлением Embox. На рис. 3 видно содержание работы: зеленым цветом обозначены те части, которые были разработаны в рамках данной курсовой работы. Оранжевым цветом обозначены сгенерированные компилятором protobuf файлы.

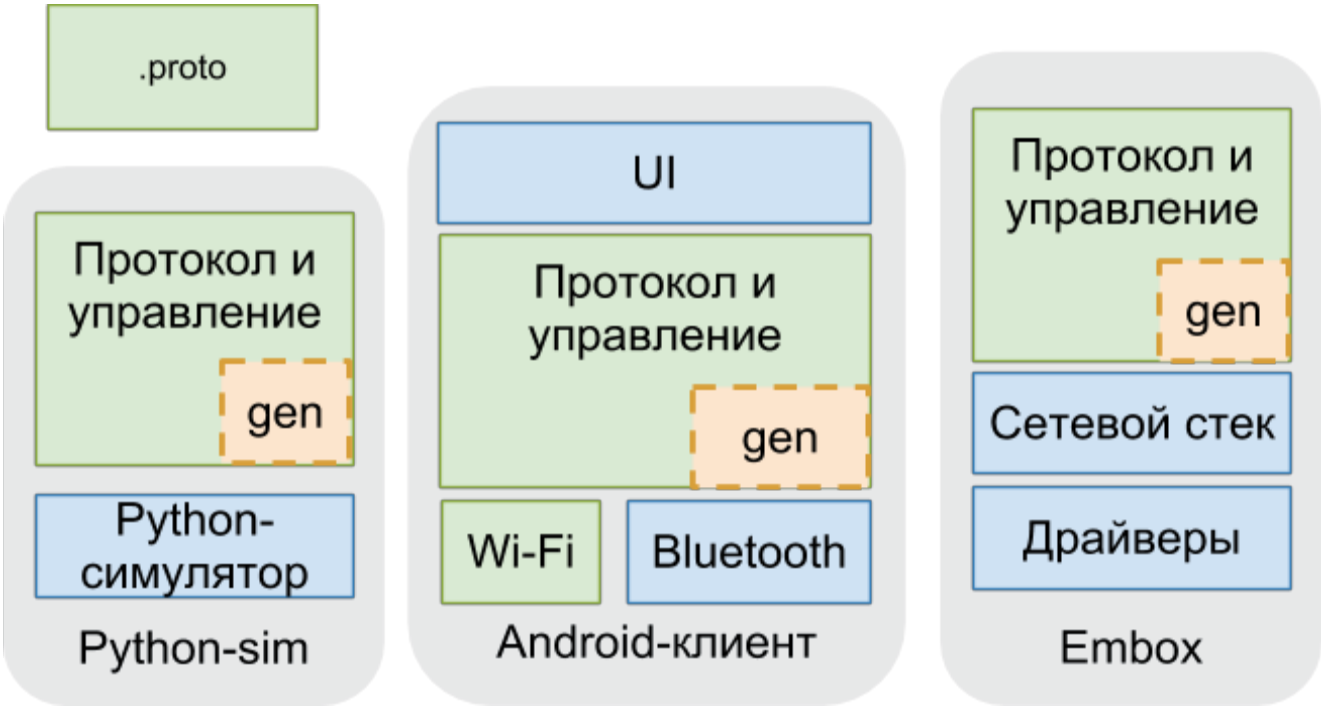

Рис. 3 Содержание работы

Android-клиент

Для управления устройством в 2011 командой Embox было разработано приложение под платформу Android. Это приложение по показателям с акселерометра управляет движением устройств, которые оно поддерживает, с помощью передачи по Bluetooth.

В ходе написания данной курсовой работы у приложения появились новые функции:

- поддержка разработанного протокола
- поддержка управления устройствами по Wi-Fi

Работоспособность приложения была протестирована на Python-симуляторе и Lego NXT на платформе Embox, о которых речь пойдет ниже.

В качестве компилятора Protocol Buffers использовался официальный компилятор для Java от Google Inc. [4]

Python-симулятор устройства

Для отладки клиента решено было создать простой симулятор на языке Python, так как несложно реализовать для симулятора основную поддержку протокола с наименьшими шансом столкнуться с непредвиденными ошибками. На сегодняшний день его функциональность невелика: симулятор представляет из себя поле, по которому перемещается мячик, управляемый Android-клиентом по Wi-Fi. Благодаря симулятору можно было полностью сосредоточиться на реализации клиента и быстро отладить его.

В качестве компилятора Protocol Buffers использовался официальный компилятор для Python от Google Inc. [4]

Устройства под управлением Embox

После того, как новый протокол был реализован на симуляторе, а клиент отлажен, была добавлена поддержка новой версии протокола для прошивки Embox для Lego NXT.

В течение года ядро Embox быстро развивалось, изменились, например, такие важные части, как подсистема ввода-ввода и планировщик потоков. Внимание команды было сосредоточено на поддержке более приоритетных платформ, таких как x86. Таким образом, на некоторое время разработка под ARM приостановилась. Помимо внедрения нового протокола необходимо было обновить конфигурацию прошивки для Lego NXT и решить связанные с этим проблемы. Их оказалось не очень много: неполадки с прерываниями и ошибка переполнения стека. В ближайшем будущем планируется добавить смену контекста для потоков для данной архитектуры и внедрить новый планировщик, что сильно улучшит эффективность работы устройства.

В качестве компилятора Protocol Buffers использовался компилятор для языка C от сторонних

разработчиков Protobuf-Embedded-C. [5]

Результаты

В ходе работы был разработан протокол, который позволяет удобно управлять соединением, позволяет устройству и клиенту проще подстраиваться под возможности друг друга. Благодаря новому протоколу внедрение поддержки новых устройств стало намного проще и удобнее. Кроме того, разработанный протокол прост как в реализации, так и в интеграции в систему, планирующей поддерживать данный протокол. Благодаря эффективному способу кодирования, используемому при сериализации сообщений Protocol Buffers, сообщения, передаваемые посредством нового протокола, не содержат избыточную информацию, что позволяет сэкономить такие ресурсы, как время и память.

Данный протокол был отлажен с помощью написанного на Python симулятора и с помощью управляющего им Android-клиента. После этого решение было реализовано для Lego NXT на платформе Embox и апробировано.

В будущем протокол можно будет расширить для передачи поточных данных, таких как аудио, видео, изображения, текстовые файлы и прочее. Кроме того, можно расширить перечень устройств и приложений, которые будут эффективно обмениваться данными по разработанному протоколу.

Список литературы

[1] USB interface tutorial covering basic fundamentals

http://www.eeherald.com/section/design-guide/esmod14.html

[2] Introduction to SIP a Beginners' Tutorial as part of Internet Multimedia

http://www.siptutorial.net/index.html

[3] Константинов Александр Сергеевич, дипломная работа “Разработка технологии взаимодействия гетерогенных систем с использованием метапрограммирования”, стр. 6-19

http://se.math.spbu.ru/SE/diploma/2011/Konstantinov%20Aleksandr%20-%20text.docx

[4] Protocol Buffers - Google's data interchange format

https://code.google.com/p/protobuf/

[5] Protobuf-Embedded-C - Protocol Buffers for Resource Constrained Embedded C Applications

https://code.google.com/p/protobuf-embedded-c/