Платформа для разработки мобильных приложений Ubiq Mobile

А.Н.Терехов, д.ф.-м.н., зав.кафедрой системного программирования

В.В.Оносовский, с.н.с. лаборатории информационных систем

Санкт-Петербургский государственный университет
Математико-механический факультет
198504, Санкт-Петербург, Университетский пр., д.28
andrey.terekhov@lanit-tercom.com , v.onossovski@gmail.com

Введение

Более двух лет группа студентов под руководством авторов этого доклада работала над реализацией платформы Ubiq Mobile[1,2]. Первоначально основной целью разработки этой платформы было облегчение разработки приложений для самых разнообразных мобильных телефонов. Дело в том, что способы отрисовки изображений, взаимодействия с радиоканалом, работы с клавиатурой и т.д. в разных телефонах сильно отличаются, поэтому разработчикам приложений приходится тратить много усилий на преодоление этих технических трудностей. Особенно много вопросов вызывает отрисовка изображений, загружаемых по дорогому и относительно медленному радиоканалу. Ubiq Mobile предлагает следующее решение: приложение исполняется в основном на сервере, а мобильный телефон играет роль тонкого клиента, связанного с сервером по специальному компактному двоичному протоколу. Были реализованы клиенты для java-телефонов, Windows Mobile и Symbian. Для этих клиентов было реализовано несколько пробных приложений, на которых было продемонстрировано, что сложность разработки приложений уменьшается в разы.

Однако стремительный рост популярности Android и iPhone и упорное нежелание пользователей дешевых телефонов покупать дополнительные сервисы снизили остроту проблемы фрагментации мобильных платформ, но задача снижения сложности разработки и расширения круга разработчиков мобильных сервисов остаётся по-прежнему актуальной. Использование платформы Ubiq Mobile позволит «обычным» программистам, не имеющим специального опыта мобильных разработок, создавать качественные мобильные сервисы, используя при этом такие возможности сервера платформы (экономия мобильного трафика, устойчивость по отношению к разрывам соединений, сохранение состояния пользовательских сессий при отсоединении мобильных клиентов и т.д.), которые с большим трудом могут быть реализованы в рамках отдельного нативного приложения. Серверная часть платформы реализована на базе .NET, причем накоплена большая библиотека процедур, используемых в различных приложениях.

С нашей точки зрения, стратегическим направлением развития платформы, радикально снижающим сложность разработки приложений, является широкое использование специализированных графических языков (DSL), позволяющих разработчикам проектировать и программировать сервисы в терминах конкретной предметной области (в качестве примеров таких предметных областей можно привести дистанционное видеонаблюдение, коллективные игры, приложения, работающие с географическими картами, информационные онлайн-сервисы и т.д.). Интересно заметить, что реализация приложения на DSL позволяет простой сменой генератора получать и прямые реализации для популярных мобильных платформ.

Платформа Ubiq Mobile

Мы предлагаем универсальную платформу для создания мобильных онлайн-сервисов, которая обеспечивает богатую функциональность, сравнимую с возможностями Mobile Ajax. В то же время её требования к ресурсам (трафик радиоканала, мощность процессора) сравнительно низкие, так что платформа может работать на широком спектре мобильных устройств и в различных сетевых окружениях, включая медленные GPRS и EDGE. Именно этим обусловлено название платформы Ubiq Mobile (от слова ubiquitous – вездесущий, повсеместный). Основная идея платформы состоит в использовании терминальной (практически, как в mainframe) архитектуры, когда все приложения исполняются на сервере, а мобильное устройство рассматривается как удаленный графический терминал. Обмен данными между серверными приложениями и мобильными клиентами осуществляется через проприетарный двоичный протокол, построенный над TCP/IP. Передача изображений осуществляется следующим образом: полностью картинка представлена только на сервере, а на мобильное устройство передается только её часть, которую видит в данный момент пользователь на своем маленьком экране. Если пользователь захочет сдвинуть изображение, скажем, вправо, сервер передаст на мобильное устройство нужный фрагмент.

Приложения исполняются на сервере, реализованном на базе Microsoft.NET, что позволяет разработчикам использовать всё богатство платформы, в частности, писать на различных языках, её поддерживающих. Приложение может практически ничего не знать о специфике устройства, которое держит в руке пользователь. С точки зрения приложения, операция вывода на экран устройства абсолютно прозрачна – приложение просто рисует графическое изображение на виртуальном холсте в памяти сервера через API платформы. Платформа автоматически поддерживает соответствие между холстом и экраном мобильного устройства через свой протокол. Пользователь может двигать окно своего экрана по холсту, используя клавиши скроллинга или джойстик устройства.

Система воспринимает все действия пользователя (нажатие клавиш, выбор конкретных контролов, т.е. управляющих клавиш экрана и т.д.) как некоторые события. Приложение обрабатывает эти события через API платформы в синхронном или асинхронном виде.

Обычно операции ввода-вывода, обеспечиваемые платформой для приложений, работающих на сервере, не зависят от конкретного устройства. Из этого правила есть всего несколько исключений, например, запрос информации о местоположении абонента. В таких случаях, если конкретное мобильное устройство не поддерживает такой функции, клиентская программа может предпринять какие-то обходные действия, например, грубо определить свое местоположение по уровню сигналов базовых станций.

Нашу платформу можно назвать «легковесной», поскольку на клиентской части исполняются только относительно простые программы, поэтому есть возможность обеспечить широкий набор мобильных устройств, включая самые дешевые. Из этого следуют и два основных ограничения платформы: относительно статичный не очень быстрый пользовательский интерфейс (по сравнению с динамическим нативным графическим интерфейсом) и невозможность работы офлайн. Но для основных классов приложений, на которые нацелена наша платформа (интерактивные информационные сервисы), эти ограничения не существенны.

Мы рассчитываем, что новая платформа – не требующая много ресурсов, легкая для программирования и для использования – расширит круг как разработчиков, так и пользователей современных мобильных сервисов и сделает новые быстрорастущие интернет-сервисы доступными новым категориям пользователей.

Рассмотрим подробнее некоторые важные аспекты реализации платформы.

Двоичный терминальный протокол

Каждая команда протокола в общем случае представляет собой дерево, элементами которого являются секции, содержащие различные по смыслу данные. Примерами секций могут быть данные, относящиеся к login-процедуре, данные, относящиеся к передаче графической информации, данные, относящиеся к управляющим элементам (control’ам) и т.д. Секции представляют собой непрерывные области данных переменной длины, последовательно размещающиеся в теле команды. В начале каждой секции в команде записывается 4-байтовый заголовок секции (Section Header), содержащий в трёх младших байтах длину секции (включая длину заголовка), а в старшем байте – код секции. Код секции определяет структуру информации, хранящейся в данной секции.

Рис. 1 Структура типового сообщения протокола

Внутри себя секция может содержать другие секции, в этом случае длина объемлющей секции будет включать в себя длины всех своих подсекций. Соответственно, подсекции могут включать в себя другие подсекции ещё более низкого уровня и т.д. Уникальность кодов секций в рамках одной команды не требуется; секции или подсекции с одинаковыми кодами могут много раз встречаться в команде. Такая структура секций удобна, в частности, для того, чтобы клиент, не поддерживающий работу с тем или иным типом данных, мог просто «пропустить» соответствующую секцию в команде без нарушения обработки остальных секций.

Данные внутри секций организованы в виде последовательностей полей фиксированной или переменной длины. Поля фиксированной длины представляют собой числа (шириной 1, 2, 3 или 4 байта), символы (шириной 2 байта, в кодировке Unicode) или байтовые поля различной длины. Числа представляются в little endian формате – т.е., вначале идут младшие байты.

Данные переменной длины начинаются с 4-байтового поля, за которым следуют собственно данные. Первые 3 байта начального поля содержат длину всей порции данных в байтах в формате little endian, а содержимое старшего байта определяет способ упаковки этих данных. В текущей версии платформы поддерживаются следующие методы упаковки:

- Данные не упакованы;
- Данные представляют собой графическую информацию, сжатую методом PNG;
- Данные представляют собой графическую информацию, сжатую методом JPEG;
- Данные представляют собой графическую информацию, сжатую с использованием wavelet-кодирования.

Важным частным случаем данных переменной длины являются строковые данные (строки), в которых сжатие не используется, а сами данные представляют собой последовательность двухбайтовых символов в кодировке Unicode.

Протокол поддерживает понятие массивов данных, которые начинаются с двухбайтового поля, содержащего количество элементов, за которым идут собственно элементы, каждый из которых может быть как фиксированной, так и переменной длины. Массивы используются, например, для единовременной передачи клиенту нескольких фрагментов изображения (массив графических данных переменной длины), для передачи текущей версии оглавления доступных сервисов (массив структур, включающих строковое имя сервиса и набор полей-атрибутов фиксированной длины) и т.д.

Описанная структура протокола сочетает компактность хранения двоичной информации с гибкостью и универсальностью, облегчающей как разбор получаемых команд , так и модификацию протокола (путём добавления новых секций и подсекций), что, в конечном счёте, позволяет существенно снизить требования приложений, разработанных в рамках платформы, к сетевым и вычислительным ресурсам.

Приложения Ubiq Mobile

Приложения, разрабатываемые для платформы Ubiq Mobile, представляют собой отдельные сборки Microsoft.NET, динамически загружаемые (deploying) на сервер Ubiq Mobile и выполняющиеся в его среде. Каждая такая сборка представляет собой библиотеку классов .NET, хотя бы один из которых должен являться наследником определяемого платформой Ubiq базового класса приложений (AbstractThreadApplication) и переопределять виртуальный метод Main (), который и вызывается системой при запуске приложения. При наличии в сборке нескольких классов – наследников AbstractThreadApplication – вызывается метод Main() самого внутреннего из них; если таких классов оказывается более одного – генерируется исключение. Каждое приложение работает в отдельном потоке с использованием пула потоков.

Приложения в системе Ubiq Mobile бывают двух типов – пользовательские приложения и приложения-сервисы. Пользовательские приложения соответствуют сессиям мобильных пользователей; приложения-сервисы создаются при старте сервера Ubiq Mobile и существуют в системе постоянно. Пользователь, подключаясь к системе с мобильного устройства, может выбрать интересующий его мобильный сервис из списка доступных сервисов , который реплицируется сервером на все мобильные Ubiq-клиенты. Во время работы пользователь может переключаться между сервисами прозрачным для них образом, подобно «щёлканию» телевизионных каналов; при переключении на новый канал (вызове другого сервиса) «старый» канал будет приостановлен (с возможностью последующего возобновления), а «новый» будет активирован.

Тот же самый механизм свёртки и возобновления используется и для сохранения состояния пользовательских сессий при разрывах мобильных соединений. При разрыве соединения соответствующее сессии пользовательское приложение продолжает существовать на сервере в «свёрнутом» виде, и при последующем входе в систему с того же самого мобильного устройства прерванная сессия будет возобновлена в том же состоянии, в каком она была прервана.

Приложения-сервисы не привязаны к пользовательским сессиям, поэтому возможность свёртки и восстановления для них не актуальна. Вместо этого, они обладают другим важным свойством: при аварийном завершении приложения-сервиса, сервер автоматически перезапускает его. Возможность многократного автоматического перезапуска необходимо учитывать при программировании приложений-сервисов.

С точки зрения пользовательского приложения обмен информацией с мобильным клиентом происходит через виртуальный холст – приложение выводит на холст графическую информацию и в асинхронном режиме получает информацию о клиентских событиях. Холст имеет многослойную структуру, в настоящее время система поддерживает 4 слоя: слой подложки, слой полутоновых изображений, слой штриховых изображений и слой управляющих элементов (controls). При передаче содержимого холста на мобильное устройство по двоичному протоколу каждый слой использует собственные методы сжатия информации, что позволяет уменьшить мобильный трафик. Кроме того, многослойность холста расширяет графические возможности системы, позволяя приложению, например, выводить текстовую информацию поверх графической «картинки».

Для доступа приложений как к внешнему миру, так и к различным частям функциональности системы платформа Ubiq Mobile предоставляет разработчикам приложений набор программных интерфейсов – API. Некоторые, самые базовые, API подключаются к приложениям по умолчанию; для использования более специфических API необходимо их явное подключение в коде приложения, которое сводится к созданию соответствующего объекта в базовом классе приложения. Приведём примеры некоторых часто используемых API.

Graphics API – основное базовое API для обмена информацией с мобильным клиентом через холст, подключаемое ко всем пользовательским приложениям по умолчанию. Представляет собой, в основном, набор объектных «переходников» к базовым функциям работы с графикой, обеспечиваемых системой Windows GDI+.

Extended Graphics API – графическое API высокого уровня, предназначенное для удобного программирования графики в пользовательских приложениях. Представляет собой библиотеку классов, поддерживающую иерархические структуры графических объектов (устройство, контейнеры, решетки (grids), управляющие элементы (controls)) подобно тому, как это реализовано, например, в библиотеке Silverlight компании Microsoft. Основное достоинство таких систем заключается в том, что они полностью берут на себя всю работу по размещению графических элементов на экране, избавляя разработчиков от необходимости вручную считать координаты.

Timer API – внутреннее API платформы, обеспечивающее основные функции работы с таймерами. Использование стандартных таймеров .NET не рекомендуется, поскольку может непредсказуемым образом интерферировать с внутренними механизмами управления приложениями, реализованными в платформе.

MessagingAPI – внутреннее API, предназначенное для обеспечения синхронного и асинхронного взаимодействия между приложениями через сообщения и почтовые ящики. Типичным примером такого взимодействия может быть «общение» пользовательского приложения, реализующего игру в «морской бой» со стороны одного

игрока (мобильного пользователя) с приложением-диспетчером, осуществляющим его коммутацию с пользовательским приложением другого игрока.

GMapAPI – это API, предназначенное для создания мобильных версий композитных сервисов (mash-ups), использующих картографическую информацию, предоставляемую сервисом Google Maps. Данное API содержит функции для получения нужного участка карты и отображения его на экране мобильного устройства, задания маркеров по географическим координатам или текстовым адресам и вывода этих маркеров на холст в виде специальных управляющих элементов системы Ubiq Mobile, что позволяет создавать интерактивные приложения (пользователь «кликает» на маркер на экране телефона, и приложение получает информацию о клиентском событии, которую может обработать в соответствии со своей логикой) и т.д. Для использования этого API на сервере должны быть запущены несколько служебных приложений-сервисов.

SNetAPI – это API для доступа приложения к различным социальным сетям (Twitter, Facebook, Вконтакте и т.д.), включающее функции авторизации пользователя и функции доступа к предоставляемой соответствующей социальной сетью пользовательской информации. Возможность использования информации, получаемой из популярных социальных сетей, значительно расширяет возможности и повышает привлекательность разрабатываемых мобильных сервисов.

Предметные области/типы приложений

1. Информационные онлайн-сервисы. С этого типа приложений, собственно, все и начиналось. Расписание электричек, автобусов, поездов, самолетов, программы театров и кино, расписание занятий – этот список можно продолжать бесконечно. Заметим, что запрашивая, скажем, расписание электричек, пользователь получит не расписание вообще, а для ближайшей станции. Сервисы этого типа могут быть и более специализированными, например, можно указать, где лежит нужный пользователю продукт в сетевом супермаркете.

2. Многопользовательские игры.
   Самый простой пример – «морской бой». Два человека играют через свои мобильные телефоны, платформа организует сессию, восстанавливает состояние после перерывов в игре, отображает фрагменты своего поля и уже известные фрагменты чужого поля, передает ходы. На этом примере особенно хорошо видно, какую большую часть рутинной работы по программированию берет на себя платформа.

3. Сервисы, отображающие информацию на карте. Представьте себе агента по продаже квартир или загородных домов. Он может показать клиенту не просто

адрес, а подробную карту, фотографии и другую информацию.
Если нужно найти ближайший ресторан, то можно показать не только адрес, но и как к нему пройти.

4. Удаленное управление «умными» устройствами. Концепция «умного дома» становится все боле популярной, SetTopBox-ы выпускаются массовыми тиражами, соответственно все больше домов и квартир оснащаются локальными сетями, объединяющими не только компьютеры, но и бытовые приборы. В недалеком будущем можно будет позвонить домой и включить видеомагнитофон на запись важного футбольного матча или спросить у холодильника, сколько банок пива осталось.

Но, шутки в сторону, сегодня уже имеется огромное количество приложений типа «спросил-ответили». Расписание электричек – это сравнительно простая задача, но для какого-нибудь мобильного банкинга одним запросом уже не обойдешься. Аутентификация, ключи обмена и доступа, задание номера счета и контрольные вопросы – это уже целая последовательность действий, подчиняющаяся определенному протоколу, со сложным автоматом управления, с необходимостью поддержания сессий. Для такого рода приложений платформа Ubiq Mobile будет существенно облегчать жизнь разработчикам приложений.

Использование графических технологий создания ПО

Мы более 25 лет применяем графические редакторы и автоматическую генерацию кода по диаграммам для создания телефонных станций и других систем реального времени [3], а также при реализации информационных систем. Начали мы еще в середине 80-ых годов с языка SDL [4], потом перешли на различные варианты UML [5]. Мы уверены, что графические диаграммы более наглядны, чем плоские тексты, облегчают общение специалистов разных профилей, радикально облегчают ввод в коллектив новых разработчиков.

Но то, что хорошо для крупных и сложных систем, не всегда подходит для относительно простых приложений. Для сложных систем принято раздельно проектировать функциональность (use case - диаграммы), структуру (классы) и поведение (диаграммы активностей). Это еще только основные типы диаграмм, UML 2.2 включает в себя более десятка типов диаграмм. Все эти диаграммы, безусловно, полезны, поскольку дают возможность взглянуть на проектируемую систему с различных точек зрения, но одновременно и привносят дополнительные трудности, например, проблемы согласования сущностей, отраженных в нескольких диаграммах.

В последние годы все большую популярность приобретает подход DSM (Domain Specific Modeling), в котором применяют специализированные графические языки (DSL), которые в одной конкретной предметной области позволяют описать алгоритм поведения проектируемой системы лаконично и наглядно. Есть исследования [6], в

которых утверждается, что применение DSM может повысить производительность разработки в 5-10 раз. Разумеется, и здесь есть свои «подводные камни», например, совершенно не очевидно, что в любой ситуации удастся придумать набор наглядных пиктограмм и ассоциаций между ними с понятной семантикой, да и реализация своей технологии для каждого DSL экономически оправдана только при массовом применении.

Уже появилось несколько метатехнологий, позволяющих в той или иной степени облегчить реализацию конкретного DSL. Задача их сравнения, определения слабых и сильных черт этих технологий выходит за рамки данного доклада. Скажем лишь, что на кафедре системного программирования СПбГУ создана графическая технология QReal [7], включающая в себя и возможности метатехнологии. Например, менее чем за 1 неделю был разработан и реализован графический язык управления роботами Lego Mindstorms NXT, причем в отличие от фирменного громоздкого средства, продающегося за большие деньги, наш вариант лежит в открытом доступе [8]. Мы решили, что и для разработки мобильных приложений стоит разработать свой DSL.

Пример приложения

Рассмотрим следующую задачу. Есть объект, охраняемый с помощью видеокамер. Пользователь хочет регулярно смотреть изображения, передаваемые с этих камер. Разумеется, передача видео слишком дорого обойдется, да, и далеко не все радиоканалы мобильных устройств с этим справятся. Но можно регулярно получать отдельные картинки, скажем, каждые 2 секунды или чуть реже.

Как все это может выглядеть? Клиент (пользователь) прочитал где-то объявление о нашем сервисе, купил камеру, установил, подключил к своему компьютеру, установил на нем драйвер камеры, сгрузив его с нашего сервера, драйвер обращается к серверу с запросом на регистрацию (RegRequest) и получает в ответ devid – номер, присвоенный сервером камере пользователя. Когда пользователь захочет посмотреть изображение, он на своем телефоне запустит бесплатно скачанную клиентскую часть платформы UbiqMobile, запустит приложение и введет devid.

Приложение видеонаблюдения состоит из множества объектов (по одному на каждое мобильное устройство), назовем их UserApplication, связанных с мобильными устройствами по нашему протоколу, и объекта, отвечающего за связь с видеокамерами по протоколу TCP/IP, назовем его Dispatcher. UserApplication и Dispatcher могут получать сообщения извне и общаться между собой внутри сервера. Мы специально подобрали не слишком тривиальный пример, чтобы показать выразительную силу DSL.

Нам понадобятся несколько глобальных переменных, описывающих состояние приложения. **Список list,** элементами которого являются тройки из целых clientid, devid и булевского flag. В этом списке содержится информация обо всех камерах, физически подключенных к системе в данный момент. Информация о текущих клиентах более

динамична, но из этого же списка можно понять, кто что смотрит. Булевский flag отражает готовность камеры к работе, он сбрасывается, если, например, кто-то выключит камеру.

Целая переменная interval определяет интервал в секундах между соседними изображениями, может принимать значения от 1 до какого-то большого числа (а по умолчанию, скажем, имеет значение 2). Посылка камере нуля означает команду stop, а натурального числа – продолжить работу.

Целая переменная devid определяет номер камеры, с которой в данный момент работает пользователь.

Целая переменная clientid определяет номер текущего мобильного устройства.

Литера c – это последняя клавиша, нажатая пользователем на телефоне.

Буфер buf содержит очередную картинку, переданную с камеры.

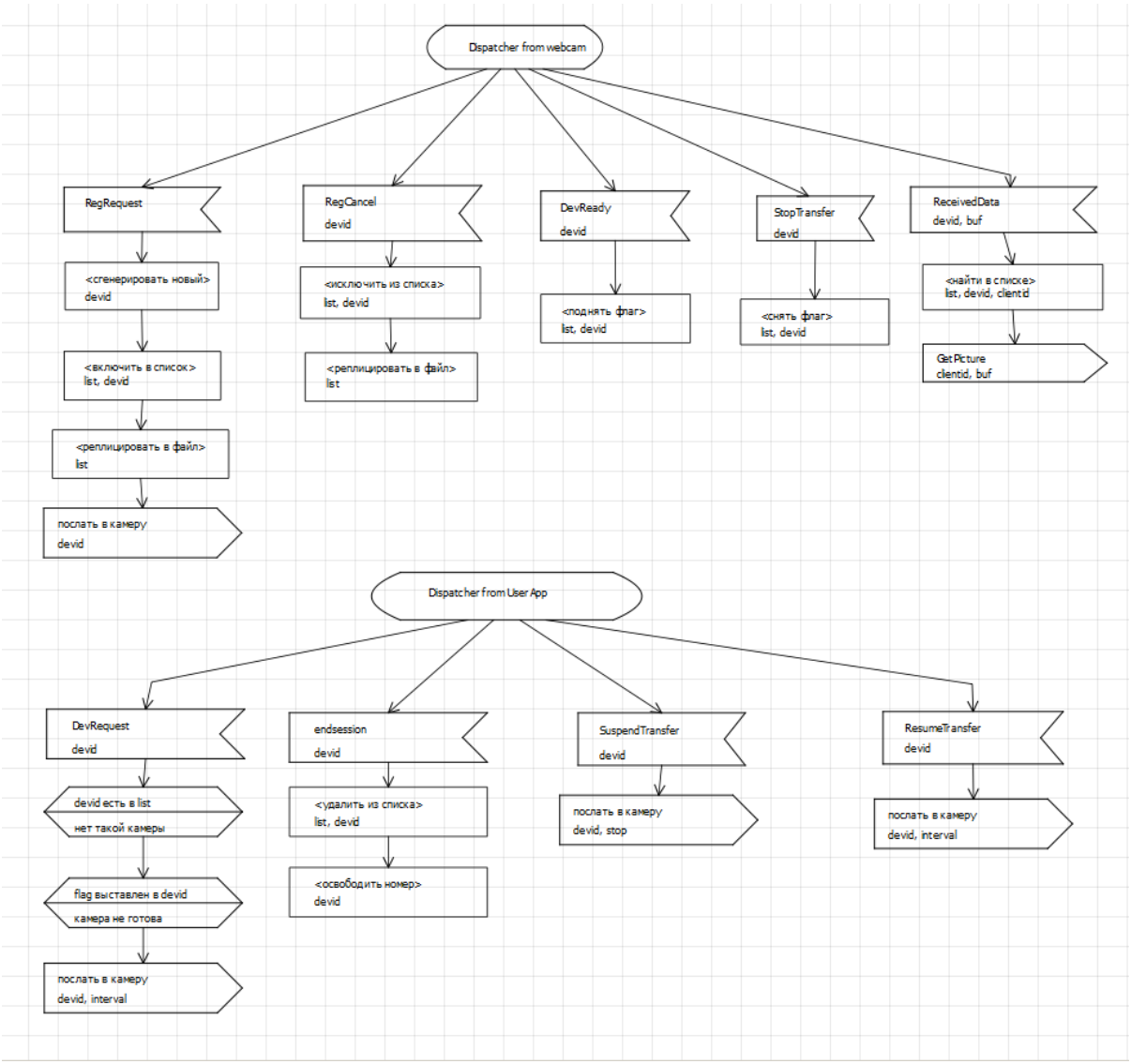

Рис. 2. Dispatcher

Описание работы приложения начнем с диспетчера (рис.2). Мы считаем, что диспетчер, в основном, находится в состоянии ожидания, в какой-то момент получает по TCP/IP одно из 5 сообщений от камер либо получает одно из 4 сообщений от UserApplication посредством внутренней службы MessagingAPI, выполняет одно или несколько действий и снова переходит в состояние ожидания.

RegRequest – требование начальной регистрации камеры, по этому сообщению генерируется очередной свободный devid (очевидно, номера камер должны переиспользоваться), этот devid заносится в список list в оперативной памяти, а затем список копируется в рабочем файле на случай перезапуска сервера, выбранный devid посылается в камеру для показа пользователю. Чтобы не множить количество графических элементов диаграмм, мы широко используем понятие стереотипа UML. Если в верхней строке символа оператора (прямоугольник) есть идентификатор в угловых скобках, то это обозначение какого-то конкретного оператора. Таких узкоспециальных операторов можно ввести сколько угодно, главное, чтобы их знал генератор кода.

RegCancel – обратное действие, камера исключается из списка обслуживаемых, а номер devid освобождается. Здесь devid является параметром сообщения.

Сообщения **DevReady** и **StopTransfer** присылаются диспетчеру тем компьютером, к которому подключена камера с номером devid. StopTransfer свидетельствует о приостановке работы камеры, DevReady – о возобновлении.

Сообщение **Received Data** сообщает, что в буфере buf готова очередная картинка с камеры devid. В списке list ищется clientid, т.е. номер клиента, который смотрит изображение с этой камеры. В UserApplication посредством MeddagingAPI посылается сообщение GetPicture с параметрами clientid и buf.

Со стороны пользователя Dispatcher получит следующие сообщения:

DevRequest – пользователь в первый раз ввел с телефона devid – номер камеры. Делаются проверки, что есть такая камера, она работоспособна (flag взведен), затем в камеру посылается значение интервала – признак начала работы.

EndSession – пользователь не хочет больше смотреть.

SuspendTransfer – пользователь просит приостановить передачу изображений или клиентская часть Ubiq Mobile вынуждена это сделать (например, временно потеряна связь с сервером).

Resume Transfer – возобновить передачу изображения.

Прежде чем говорить об абонентском приложении, опишем одну подпрограмму.

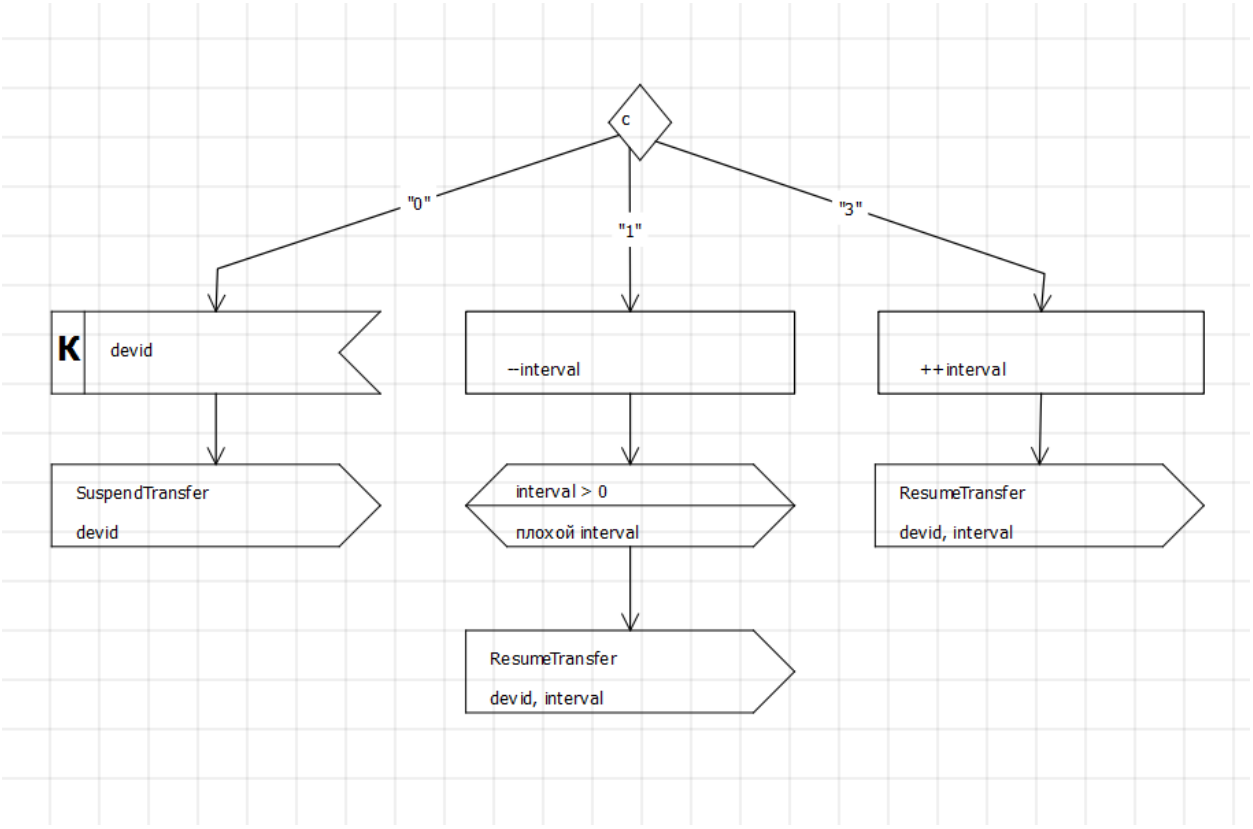

Рис. 3. Подпрограмма UserAction

UserAction (рис.3) описывает реакции системы в ответ на действия пользователя. Нажатие клавиши «Ø» означает, что пользователь хочет перейти на просмотр другой камеры, «1» - уменьшить интервал между картинками, а «3» – увеличить. После разработки вручную нескольких приложений мы заметили, что проверок приходится программировать на удивление много. Поэтому был придуман специальный оператор – «гробик», в верхней части которого пишется условие, а в нижней – текст сообщения, которое выводится на телефон пользователя, если условие не выполняется.

Оператор со стереотипом <подпрограмма> означает вызов подпрограммы с именем, указанным в нижней строчке. Оператор без всякого стереотипа – просто фрагмент текста на используемом языке программирования (чаще всего, C#).

Наконец, **Приложение пользователя** (рис.4). Оператор <литерально напечатать> означает просто вывод на экран телефона одной или нескольких строк. Шрифт, цвет, фон и другие технические детали – это атрибуты, которые можно задать в специальном окне графического редактора, но в большинстве случаев используются значения по умолчанию, так что этот оператор экономит массу усилий разработчика приложений.

Если в левой части оператора приема сообщений указана буква К (клавиатура или keyboard), то это обозначение ожидания ввода с клавиатуры. Если тип параметра литерный, то ожидается ввод одной литеры, если целый, то целого числа.

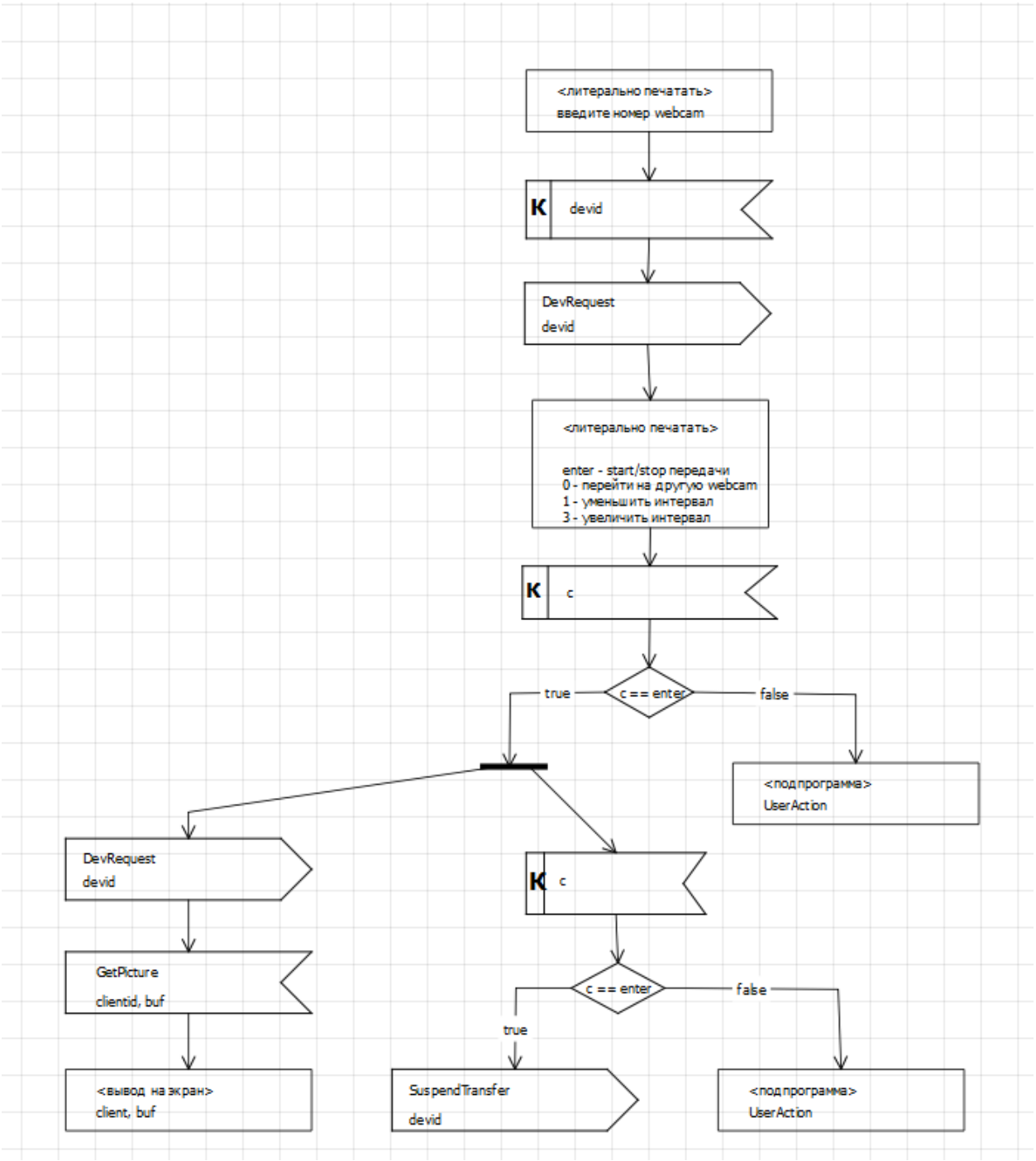

Рис. 4. UserApplication

Ромбик обозначает условный оператор, внутри которого пишется проверяемое условие, а от боковых углов отходят две линии, обозначенные символами **true** и **false**. Этот же ромбик может обозначать оператор выбора (**case**), тогда от двух нижних ребер (а не углов!) может отходить две или более линии, помеченные какими-то значениями (целыми или литерными).

Увидев на экране телефона основное меню, пользователь может сразу нажать клавишу enter и получать изображения. Однако, параллельно с этим (обратите внимание

на графический символ распараллеливания) система ожидает от пользователя команды на смену камеры или величины интервала.

В частности, повторное нажатие клавиши enter в процессе передачи картинок означает приостановку передачи изображений. UserApplication не имеет прямого доступа к камерам, поэтому посылает сообщение SuspendTransfer в Dispatcher. Аналогично надо поступить с ResumeTransfer.

Заключение

DSM-подход, несмотря на его несомненные достоинства [6], в силу различных причин не получил достаточно широкого распространения в сфере «традиционной» разработки промышленного ПО. Мы полагаем, что в области разработки мобильных приложений и сервисов ситуация принципиально иная, прежде всего, из-за более высокой сложности разработки мобильных сервисов по *сравнению* с «традиционным» ПО и, соответственно, более строгих требований к квалификации разработчиков. Кроме того, мобильные приложения и сервисы более узкоспециализированы и разделены по предметным областям. Таким образом, использование DSM-подхода при разработке мобильных сервисов позволит, во-первых, существенно снизить порог сложности программирования и расширить круг потенциальных разработчиков мобильных приложений, а, во-вторых, естественная специализация мобильных сервисов и их разделение по нишам существенно облегчит разработку самих DSL, ориентированных на тот или иной класс приложений. Платформа Ubiq Mobile, используемая в качестве run-time платформы для DSL, обеспечивает высокий уровень производительности и надёжности создаваемых приложений, а структура API платформы Ubiq позволяет существенно упростить генерацию кода приложений из диаграмм состояний DSL по сравнению, например, с генерацией нативного кода для мобильных платформ Symbian или Android. Мы убеждены, что сочетание DSM-подхода с использованием высокоэффективной платформы Ubiq Mobile даст качественно новый результат, повысив доступность современных мобильных сервисов как с точки зрения их разработки «обычными» программистами, так и с точки зрения доступности для широкого круга «некомпьютеризованных» массовых пользователей.

Литература

[1] V. Onossovski, A.Terekhov, Ubiq Mobile – a New Universal Platform for Mobile Online Services, Proceedings of 6th seminar of FRUCT Program, 2009

[2] V. Onossovski, A.Terekhov, Modern Interactive Internet Services, Proceedings of 7th Conference of Open Innovations Framework Program FRUCT, 2010

[3] Терехов А., Долгов П., Иванов А., Кознов Дм., Лебедев А., Мурашова Т., Парфенов В., "Объектно-ориентированное расширение технологии RTST", Сб. “Записки семинара

кафедры системного программирования "CASE-средства RTST++". Вып.1, СПб, Издательство СПбГУ, 19

[4] http://www.itu.int/ITU-T/studygroups/com10/languages/Z.100_1199.pdf

[5] http://www.uml.org/

[6] Kelly, S., Tolvanen, J.-P. Domain-Specific Modeling: Enabling Full Code Generation // Wiley-IEEE Computer Society Press. 2008. 448 pp.

[7] Терехов А.Н., Брыксин Т.А., Литвинов Ю.В., Смирнов К.К., Никандров Г.А., Иванов В.Ю., Такун Е.И., Архитектура среды визуального моделирования QReal, Системное программирование, вып. 4, 2009

[8] http://se.math.spbu.ru/SE/qreal