МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РОССИЙСКОЙ ФЕДЕРАЦИИ
(МИНОБРНАУКИ)

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ «САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ»
(СПбГУ ИТМО)

УТВЕРЖДАЮ
Ректор СПбГУ ИТМО,
докт. техн. наук, профессор
В. Н. Васильев

____ _____ 2006 г.

Подп. и дата	
Инв. № дубл.	
Взам. инв. №	
Подп. и дата	
Инв. № подл.	

ТЕХНОЛОГИЯ АВТОМАТНОГО ПРОГРАММИРОВАНИЯ: ПРИМЕНЕНИЕ И ИНСТРУМЕНТАЛЬНЫЕ СРЕДСТВА

МЕТОДИЧЕСКИЕ РЕКОМЕНДАЦИИ И УКАЗАНИЯ ПО РАЗРАБОТКЕ, БАЗИРУЮЩИЕСЯ НА ПРИНЦИПАХ АВТОМАТНОГО ПРОГРАММИРОВАНИЯ

ЧАСТЬ 3. МОБИЛЬНЫЕ СИСТЕМЫ

Листов 27

Декан факультета «Информационных технологий и программирования»
докт. техн. наук, профессор
_____В.Г. Парфенов

Руководитель темы
заведующий кафедрой «Технологий программирования»,
докт. техн. наук, профессор
_____ А.А. Шалыто

Ответственный исполнитель
доцент кафедры «Технологий программирования», канд. физ.-мат. наук
_____Ф.А. Новиков

2006

СПИСОК ОСНОВНЫХ ИСПОЛНИТЕЛЕЙ

Руководитель темы д.т.н., профессор	А.А. Шалыто
Ответственный исполнитель к.ф.-м.н., с.н.с.	Ф.А. Новиков
Ведущие исполнители:	
Аспирант	В.С. Гуров
Магистрант	М.А. Мазин
В работе принимали участие:	
Ведущий научный сотрудник, д.т.н., профессор	В.В. Антипов
Ведущий научный сотрудник, к.т.н.	В.В. Киселев
Ведущий научный сотрудник, к.т.н.	Р.Н. Котляр
Ведущий научный сотрудник, к.т.н.	Ю.П. Московцев
Ведущий научный сотрудник, к.т.н., доцент	В.А. Третьяков
Ведущий научный сотрудник, к.т.н.	Г.М. Файкин
Ассистент, к.т.н.	Д.Г. Шопырин
Аспирант	И.М. Аничкин
Аспирант	М.А. Казаков
Аспирант	Г.А. Корнеев
Аспирант	П.Г. Лобанов
Ведущий инженер	К.В. Вавилов
Магистрант	М.А. Коротков
Магистрант	А.П. Лукьянова
Студент	Н.И. Поликарпова
Студент	Б.М. Ярцев

АННОТАЦИЯ

Описывается методика разработки приложений для мобильных устройств на основе автоматного подхода, разработанная в процессе проведения опытно-конструкторских работ по **теме (Шифр ИТ-13.4/004): «Технология автоматного программирования: применение и инструментальные средства» (VI очередь),** выполняемой в рамках Федеральной целевой научно-технической программы «Исследования и разработки по приоритетным направлениям развития науки и техники» на 2002-2006 годы по государственному контракту № 02. 435.11.1009 от 01.08.2005, заключенному между Федеральным агентством по науке и инновациям и Государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на основании решения Конкурсной комиссии Роснауки № 3 (протокол от 01 августа 2005 г. № 17).

Приведено общее описание методики, описаны основные понятия, элементы и положения автоматного программирования для рассматриваемого класса задач.

Описаны четыре этапа методики: постановка задачи, технический дизайн, создание статической и динамической моделей, создание кода, тестирование. Применение методики описано по шагам на примере создания автоответчика для телефона *Nokia 6600*.

СОДЕРЖАНИЕ

1. ВВЕДЕНИЕ

Настоящая методика разработана в процессе проведения опытно-конструкторских работ по **теме (Шифр ИТ-13.4/004): «Технология автоматного программирования: применение и инструментальные средства» (VI очередь),** выполняемой в рамках Федеральной целевой научно-технической программы «Исследования и разработки по приоритетным направлениям развития науки и техники» на 2002-2006 годы по государственному контракту № 02. 435.11.1009 от 01.08.2005, заключенному между Федеральным агентством по науке и инновациям и Государственным образовательным учреждением высшего профессионального образования «Санкт-Петербургский государственный университет информационных технологий, механики и оптики» на основании решения Конкурсной комиссии Роснауки № 3 (протокол от 01 августа 2005 г. № 17).

Отличие данной методики от общепринятых методик объектно-ориентированного анализа и проектирования [1] состоит в том, что она определяет формальный подход к проектированию динамических аспектов программного обеспечения мобильных систем, а также задает правила перехода от динамической модели системы к коду на целевом языке программирования.

2. НАЗНАЧЕНИЕ, ОБЛАСТЬ ПРИМЕНЕНИЯ И ОСНОВНЫЕ ПОЛОЖЕНИЯ МЕТОДИКИ

Данная методика описывает процесс создания *C++* приложений для мобильных устройств с ограниченными ресурсами на основе автоматного подхода для платформы *Symbian* (http://www.symbian.com).

2.1. Место методики в цикле разработки программного обеспечения

В промышленном программировании при создании программных систем обычно выделяют этапы:

- **Постановка задачи**. Включает в себя сбор требований и создание прототипа программы.
- **Технический дизайн**. Состоит в создании на основе собранных требований технического задания, описывающего модель системы с помощью диаграмм, псевдокода и поясняющего текста.
- **Создание кода**. Состоит в реализации системы для целевой программно-аппаратной платформы в соответствии с техническим дизайном.
- **Тестирование**. Завершает цикл разработки и представляет собой отладку созданного приложения и проверку соответствия реализации собранным требованиям.

Семантический разрыв при передаче знаний между этапом технического дизайна и этапом написания кода заключается в том, что разработчик реализует систему в соответствии со своим пониманием технического задания.

Признание этого факта привело к появлению такого направления в программной инженерии как «проектирование на базе моделей» (*Model-Driven Architecture*) [2]. Основной идеей этого подхода является независимое рассмотрение моделей, создаваемых при проектировании системы, от деталей их реализации на конкретной программно-аппаратной платформе.

В подходе *MDA* модели программных систем представляются с помощью «Унифицированного языка моделирования» (*Unified Modeling Language, UML*) [3].

Если в течение ряда лет этот язык использовался только для представления моделей, то в последнее время все большую популярность приобретает идея *исполняемого UML* [4]. Это связано с тем, что практическое использование *UML* в большинстве случаев ограничивается только моделированием статической части программ с помощью диаграмм классов и генерацией по ним каркаса кода программы, а моделирование динамических аспектов программ затруднено в связи с отсутствием в стандарте формального и однозначного описания правил интерпретации (операционной семантики) поведенческих диаграмм.

В работе [5] предложен метод проектирования программ с явным выделением состояний, названный «*SWITCH*-технология» или «автоматное программирование». В дальнейшем этот подход был развит для событийных систем [6], а потом и для объектно-ориентированных [7].

Особенность этого подхода состоит в том, что программы предлагается строить также, как осуществляется автоматизация технологических процессов, в ходе которой первоначально строится схема связей, содержащая источники информации, систему управления, объекты управления и обрат-

ные связи от объектов к системе управления. В предлагаемом подходе система управления реализуется в виде системы взаимодействующих конечных автоматов, каждый из которых имеет несколько входов и выходов

SWITCH-технология определяет для каждого автомата два типа диаграмм (схему связей и граф переходов) и их операционную семантику. При наличии нескольких автоматов строится схема их взаимодействия. Для этих диаграмм была предложена соответствующая нотация (http://is.ifmo.ru/?i0=science&i1=minvuz2).

Предлагается, сохранив автоматный подход, использовать *UML*-нотацию при построении диаграмм в рамках *SWITCH*-технологии. При этом, используя нотацию *UML*-диаграмм классов, строится схема связей автоматов, определяющая их интерфейс, а графы переходов автоматов строятся с помощью нотации *UML*-диаграммы состояний. При наличии нескольких автоматов их схема взаимодействия не создается, а они изображаются на диаграмме классов. Диаграмма классов (как схема связей) и диаграммы состояний образуют предлагаемый графический язык.

Описываемая в настоящей работе методика затрагивает все фазы разработки программного обеспечения. При этом сбор и документирование требований, а также часть технического дизайна (построение статической модели системы), выполняются стандартными для объектного подхода методами [1]. Создание динамической модели системы и кода основаны на использовании автоматного подхода, что позволяет избежать упомянутого выше семантического разрыва за счет прозрачного перехода от моделей системы к ее реализации.

2.2. Основные положения методики

Для проектирования программ для мобильных устройств на основе автоматного подхода предлагается следующая пошаговая методика.

1. Постановка задачи.
    - В виде неформального текстового описания задается набор требований к программе.
    - Создается прототип приложения в виде набора экранных форм пользовательского интерфейса. Диаграмма переходов между экранными формами задается в виде диаграммы состояний *UML*. Данная диаграмма является основой для создания динамической модели системы, так как описывает поведение системы с точки зрения конечного пользователя.
2. Технический дизайн.

    2.1. Создание статической модели системы.
    - На основе анализа требований разрабатывается статическая модель системы, определяющая сущности и отношения между ними.

    2.2. Создание динамической модели системы.
    - В отличие от традиционных для объектно-ориентированного программирования подходов, в предлагаемом подходе сущности делятся на источники событий, объекты управления и автоматы. Источники событий активны — они по собственной инициативе воздействуют на автоматы. Объекты управления пассивны — они выполняют действия по командам от автоматов. Объекты управления также могут формировать значения входных переменных для автоматов. Автоматы активируются источниками событий и

на основании значений входных переменных и текущего состояния воздействуют на объекты управления, переходя в новое состояние.

- Используя нотацию диаграммы классов, строится схема связей автоматов, задающая интерфейс каждого их них. На этой схеме слева отображаются источники событий, в центре — автоматы, а справа — объекты управления. Источники событий с помощью *UML*-ассоциаций связываются с автоматами, которым они поставляют события. Автоматы связываются с объектами, которыми они управляют. Схема связей обычно также является и схемой взаимодействия автоматов.

- Схема связей, кроме задания интерфейсов автоматов, выполняет функцию, характерную для диаграммы классов — задает объектно-ориентированную структуру программы.

- Каждый объект управления содержит два типа методов, реализующих входные переменные (`xj`) и выходные воздействия (`zk`).

- Для каждого автомата с помощью нотации диаграммы состояний строится граф переходов типа *Мура-Мили*, в котором дуги могут быть помечены событием (`ei`), булевой формулой из входных переменных и, при необходимости, формируемыми на переходах выходными воздействиями. Дуги могут также помечаться именами вызываемых автоматов с указанием соответствующих событий. В вершинах могут указываться выходные воздействия и имена вложенных автоматов. При этом вложенность автоматов, в отличие от вызываемости, на схеме связей не отражается. Каждый автомат имеет одно начальное и произвольное количество конечных состояний.

- Состояния на графе переходов могут быть простыми и сложными. Если в состояние вложено другое состояние, то оно называется сложным. В противном случае состояние простое. Основной особенностью сложных состояний является то, что наличие дуги, исходящей из такого состояния, заменяет однотипные дуги из каждого вложенного состояния.

- Все сложные состояния неустойчивы, а все простые, за исключением начального – устойчивы. При наличии сложных состояний в автомате, появление события может привести к выполнению более одного перехода. Это происходит в связи с тем, что сложное состояние является неустойчивым и автомат выполняет переходы до тех пор, пока не достигнет первого из простых (устойчивых) состояний. Отметим, что если в графе переходов сложные состояния отсутствуют, то, как и в *SWITCH*-технологии, при каждом запуске автомата выполняется не более одного перехода.

- Выполняется проверка корректности построенной динамической модели. При этом проверяются следующие свойства: достижимость всех состояний на графах переходов, полнота и непротиворечивость множеств переходов для каждой вершины каждого графа переходов.

3. Создание кода.

    - Каждая входная переменная и каждое выходное воздействие являются методами соответствующего объекта управления, которые реализуются вручную на целевом языке программирования. Источники событий также реализуются вручную.

    - Использование символьных обозначений в графах переходов позволяет весьма компактно описывать сложное поведение проектируемых систем. Смысл таких символов

задает схема связей. При наведении курсора на соответствующий символ на графе переходов во всплывающей подсказке отображается его текстовое описание.

- После создания модели программы выполняется генерация *C++* кода для платформы *Symbian* [8], который впоследствии компилируется и запускается. Сгенерированный код содержит встроенные методы для вывода на консоль протокола работы.

4. Тестирование

- Отладка кода, сгенерированного по автоматной модели, выполняется с помощью анализа протокола работы. В случае обнаружения некорректного поведения модели, изменения вносятся в модель. После этого выполняется перегенерация кода.
- Отладка кода, написанного вручную, осуществляется стандартными методами.

Предлагаемая методика, в отличие от других известных открытых средств, позволяет спроектировать программу в целом.

2.3. Инструментальное средство для поддержки методики

Для поддержки описанной методики создано инструментальное средство *UniMod* [9–11]. Это средство ориентировано на разработку автоматных *Java*-приложений. При этом реализованы два варианта запуска автоматных моделей: прямая интерпретация модели и генерация *Java*-кода из модели. Генерация кода выполняется с помощью *Velocity* шаблонов (http://jakarta.apache.org/velocity/).

Для языка *C++* также были созданы шаблоны для генерации кода, что позволило применять пакет *UniMod* при использовании этого языка в качестве целевого языка программирования. Прямая интерпретация автоматной модели при использовании языка *C++* не реализована, так как интерпретационный подход является более медленным и ресурсоемким по сравнению с компилятивным подходом, и поэтому его использование нецелесообразно для мобильных устройств.

Пакет *UniMod* позволяет автоматизировать создание динамической модели системы и генерацию кода по модели. При этом автоматически выполняется проверка корректности построенной модели. В случае нахождения ошибок пользователю предлагаются альтернативные пути их устранения. В дальнейшем производится интеграция части программы, построенной автоматически, с фрагментами кода, написанными вручную. Это обеспечивает построение программы в целом.

Для разработки *C++* приложений с использованием пакета *UniMod* необходимо для платформы *Symbian* установить следующее программное обеспечение:

- *Eclipse SDK* 3.1.2 (http://www.eclipse.org)
- *Eclipse C/C++ Development Tools (CDT)* 3.0.2 (http://www.eclipse.org/cdt/)
- *UniMod* 1.3.1.36 (http://unimod.sourceforge.net/)
- *Symbian OS development plug-in for Eclipse* (http://www.newlc.com/Eclipse-plug-in-for-Symbian-C.html)
- *Active Perl* (http://www.activestate.com/Products/ActivePerl/)
- *S60 Platform SDK for Symbain OS, for C++* (http://www.forum.nokia.com/)
- *Microsoft .NET Software Development Kit* (http://www.microsoft.com/downloads/)
- *Microsoft Visual C++ Toolkit 2003* (http://www.microsoft.com/downloads/)

- *Microsoft Windows Platform SDK* (http://www.microsoft.com/downloads/)

Опишем цикл разработки программного обеспечения для рассматриваемого класса устройств при использовании указанного инструментального средства.

1. Создать *Java*-проект.
2. Создать в этом проекте *UniMod*-модель.
3. Сгенерировать классы «заглушки» для объектов управления и источников событий. При этом будет сгенерирован код, реализующий автоматы из *UniMod*-модели на языке *Symbian C++*.
4. Создать *Symbian OS*-проект.
5. Перенести в этот проект код сгенерированный на шаге 3.
6. Реализовать на языке *Symbian C++* объекты управления, источники событий и вспомогательные классы для инициализации приложения, создания экземпляров объектов управления, поставщиков событий и сгенерированных автоматных классов.

Для *UniMod*-модели генерируются пара файлов: заголовочный *Symbian C++* файл и файл, содержащий реализацию методов классов, объявленных в заголовочном файле. Для каждой автоматной модели генерируется несколько классов:

- класс, реализующий «движок» автоматный модели, который является интерфейсом между автоматной частью программы, сгенерированной инструментальным средством *UniMod* и частью программы, которую необходимо запрограммировать вручную;
- вспомогательный базовый автоматный класс, содержащий поля и методы, общие для всех автоматов модели;
- по классу на каждый автомат, входящий в модель.

Класс, сгенерированный для одного автомата, модели содержит:

- конструктор, принимающий на вход экземпляр класса «движка» модели;
- метод `HandleL`, который реализует логику обработки событий;
- методы для установки ссылок на экземпляры объектов управления и вызываемых автоматов;
- перечисление (`enum`) всех состояний, объявленных в автомате;
- поле, соответствующее переменной состояния;
- ссылки на все связанные с автоматом объекты управления и вызываемые автоматы;
- метод `IsStable`, вычисляющий является ли состояние, переданное ему в качестве формального параметра, устойчивым;
- метод `GetSuperstate`, вычисляющий состояние, в которое вложено состояние, переданное ему в качестве формального параметра;
- метод `TransiteOnEventL`, реализующий переход из текущего активного состояния, в зависимости от события и значений входных переменных;

- метод `TransiteToStable`, реализующий переход из состояния заданного входным параметром в соответствующее ему устойчивое состояние;
- метод `EnterStateL`, реализующий вход автомата в состояние;
- метод `InvokeSubmachineL`, реализующий вызов автоматов, вложенных в заданное состояние.

Объекты управления и поставщики событий должны быть запрограммированы вручную. Класс, реализующий «движок» автоматной модели, содержит методы двухфазной инициализации [8], которые принимают на вход ссылки на экземпляры объектов управления, и инициализируют автоматы модели.

Для посылки события автоматам модели, поставщики событий должны вызывать метод «движка» автоматной модели `HandleL`, который, в свою очередь, делегирует вызов головному автомату модели. Поставщики событий могут передать параметры события, используя контекст, ссылку на который они могут получить у «движка» автоматной модели, используя метод `Context`.

Для того чтобы запустить автоматную модель требуется вручную запрограммировать создание объектов управления, поставщиков событий и «движка» автоматной модели.

При создании «движка» автоматной модели, ему передаются ссылки на объекты управления.

Предполагается, что поставщики событий запрограммированы таким образом, чтобы иметь возможность подписываться и обрабатывать системные события, а затем транслировать их в вызовы метода `HanldeL` «движка» автоматной модели. Так как в приложениях на *Symbian C++* используется невытисняющая многозадачность, проблемы с одновременной обработкой автоматом нескольких событий не возникает – новое событие начинает обрабатываться только по завершении обработки текущего.

3. ПРИМЕР ПРИМЕНЕНИЯ МЕТОДИКИ

В качестве примера применения методики и инструментального средства рассматривается разработка автоответчика для мобильного телефона Nokia 6600.

3.1. Постановка задачи

Опишем требования.

1. Разработать программу-автоответчик для мобильного телефона Nokia 6600.
2. Программа должны быть написана на языке *C++* для платформы *Symbian.*
3. Программа должна поддерживать два режима работы: режим настройки голосовых сообщений и режим ответа на звонки.
4. В режиме настройки голосовых сообщений программа должна позволять пользователю записать произвольное количество голосовых сообщений для определенного абонента или группы абонентов из встроенной в телефон телефонной книги.
5. В режиме ответа на звонки программа должна автоматически снимать трубку, определять номер абонента и проигрывать предназначенное для этого абонента сообщение.

На рис. 1–5 показан прототип приложения в виде экранных форм. Каждый рисунок состоит из двух частей: слева показана экранная форма, а справа – контекстное меню, определяющее набор возможных действий.

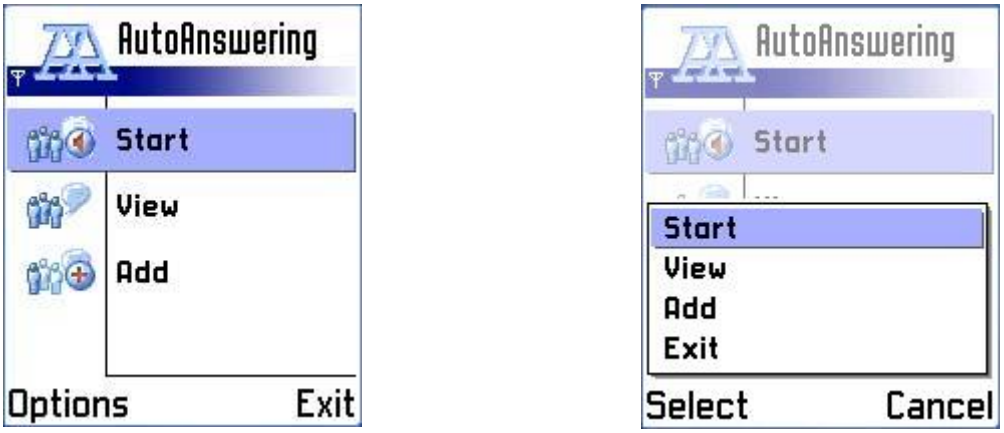

Рис. 1. Главная экранная форма (*Main*)

Рис. 2. Форма выбора абонента или группы абонентов (*Select user or group*)

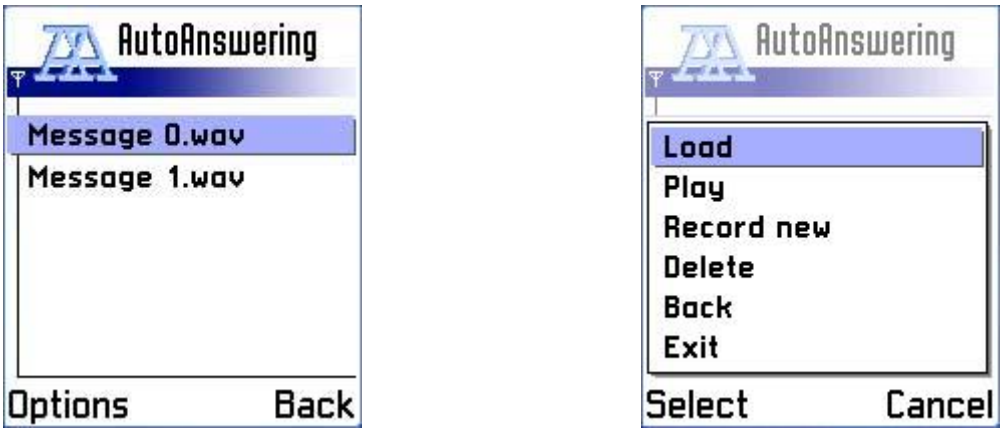

Рис. 3. Форма выбора записанного сообщения (*Select greeting message*)

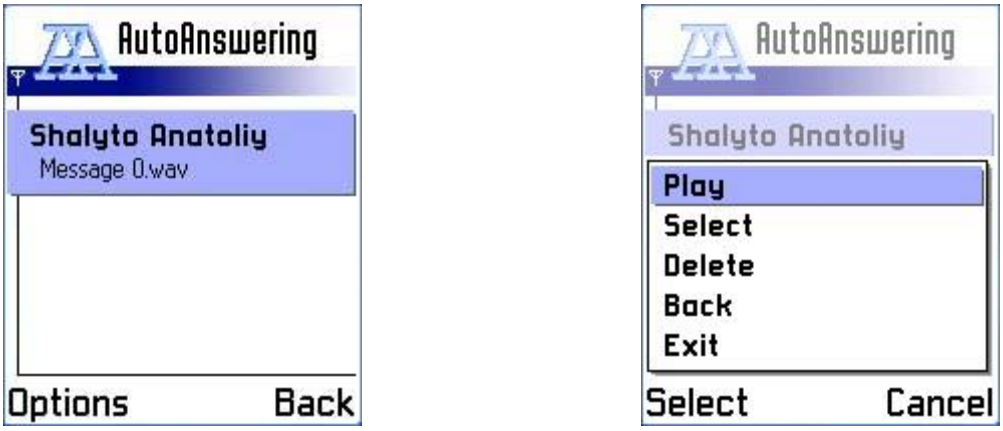

Рис. 4. Форма просмотра абонентов и предназначенных для них сообщений (*View*)

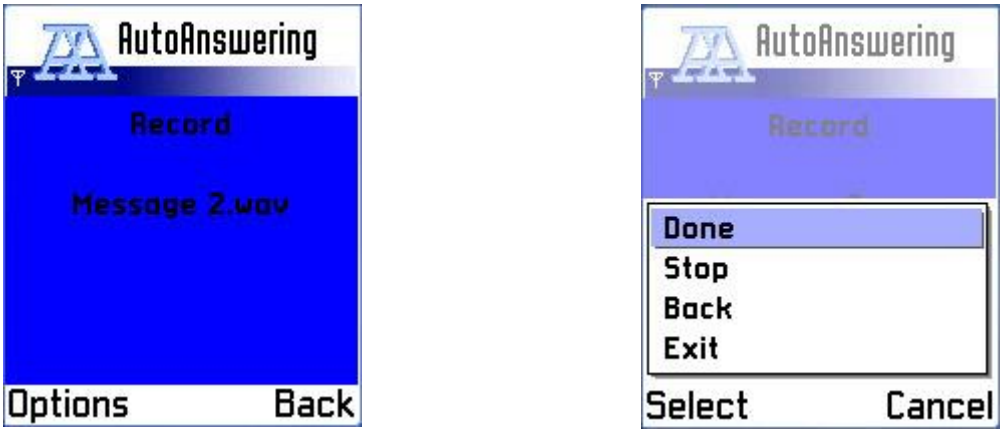

Рис. 5. Форма записи нового сообщения (*Record*).

На рис. 6 представлена диаграмма состояний, описывающая переходы между экранными формами. Названия состояний соответствуют названиям экранных форм в подписях под рис. 1–5. Для состояния *Active* экранная форма отсутствует, так как в этом состоянии приложение переходит в фоновый режим ответа на звонки.

События на переходах на рис. 6 соответствуют пунктам контекстного меню на рис. 1–5.

Составное состояние *Application Running* введено того, чтобы сделать диаграмму более компактной, так как оно позволяет только один раз определить переход в финальное состояние, который возможен из любого вложенного состояния.

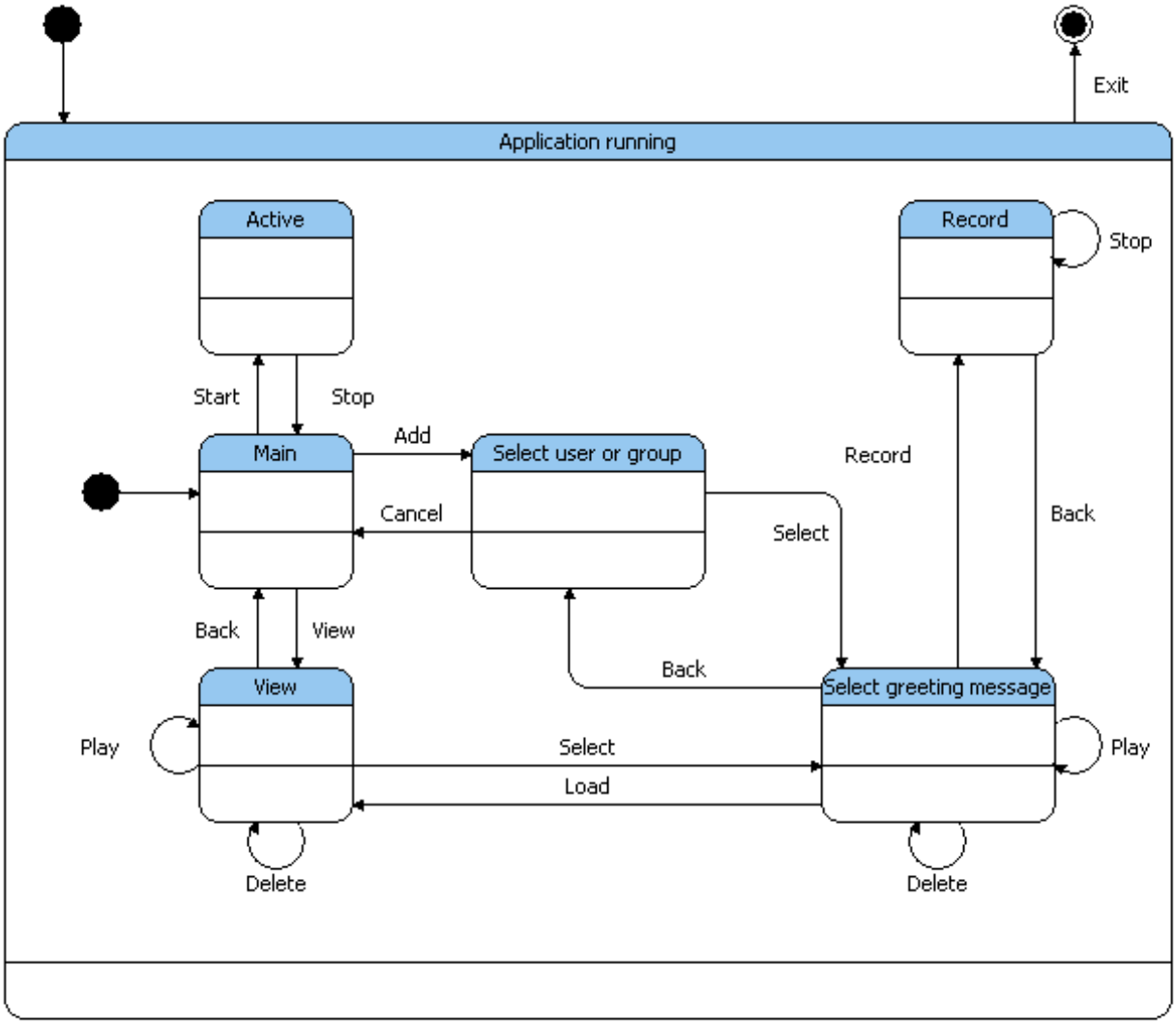

Рис. 6. Диаграмма состояний, описывающая переходы между экранными формами

3.2. Технический дизайн

3.2.1. Статическая модель системы

На рис. 7 представлена статическая модель системы. На ней с помощью стереотипов <<*event-provider*>>, <<*controlledobject*>> и <<*statemachine*>> выделены источники событий, объекты управления и автоматы соответственно.

Стереотипом <<*datatype*>> помечены классы, не обладающие поведением и являющиеся контейнером для данных.

Стереотипом <<*service*>> помечены классы, являющиеся составной частью операционной системы *Symbian* и предоставляющие приложению различные сервисные функции для взаимодействия с окружающей средой.

Классы, не помеченные стереотипами, показаны для удобства представления предметной области и не имеют соответствующей реализации. Так класс *Phone* логически агрегирует в себе все сер-

висы, предоставляемые запущенным на нем приложениям, а класс *User* соответствует пользователю, работающему с телефоном.

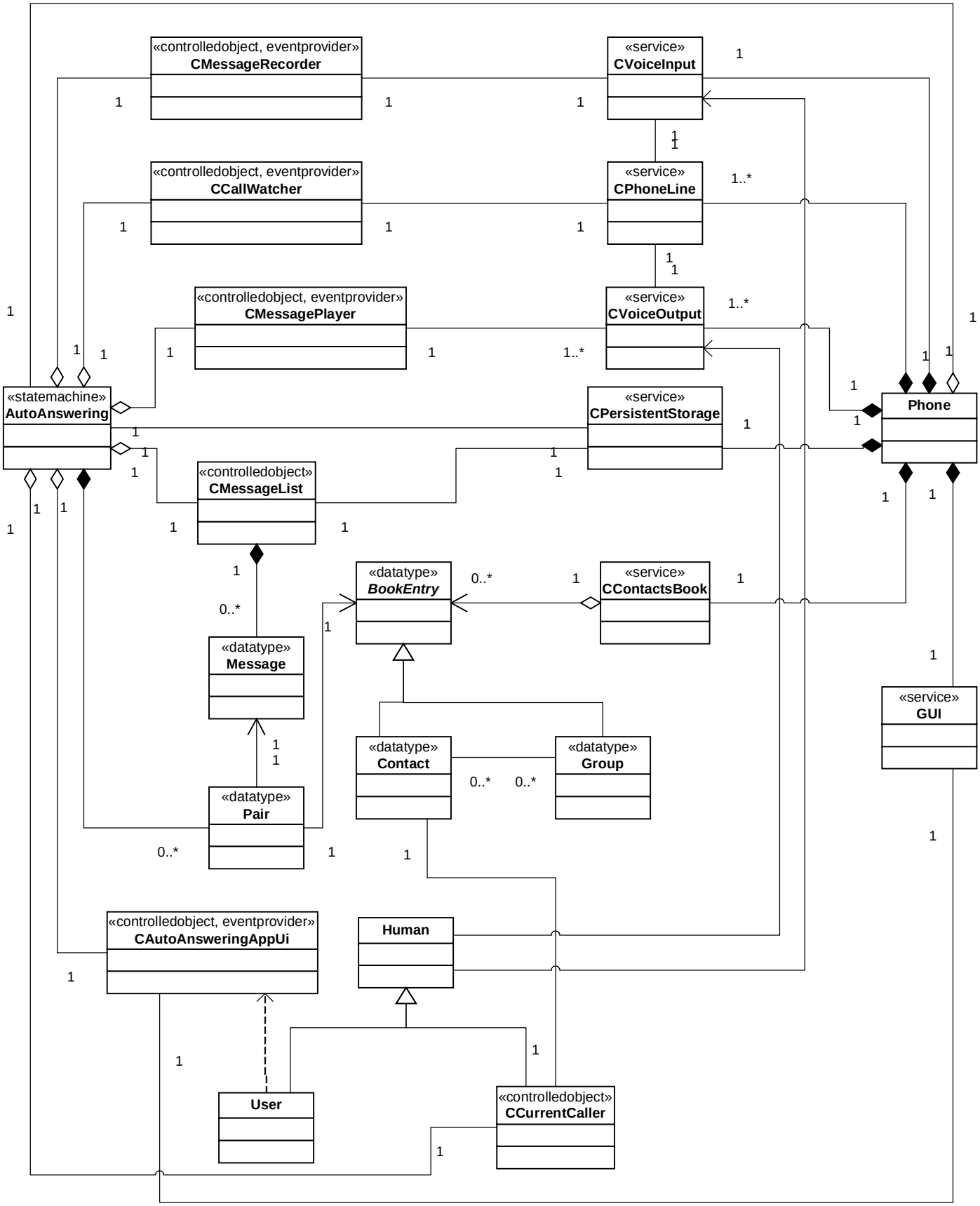

Рис. 7. Статическая модель системы

3.2.2. Динамическая модель системы

На рис. 8 представлена схема связей автоматов. Слева на ней расположены источники событий, а справа – автоматы и объекты управления. Отметим, что такие классы статической модели системы, как *CAutoAnsweringAppUi*, *CMessagePlayer*, *CMessageRecorder*, *CCallWatcher*, *CCurrentCaller* в динамической модели играют роль и источников событий и объектов управления. Поэтому на схеме связей они показаны дважды.

Классы со стереотипами <<*datatype*>> и <<*service*>> не показаны на схеме связей, так как они не являются частью динамической модели системы.

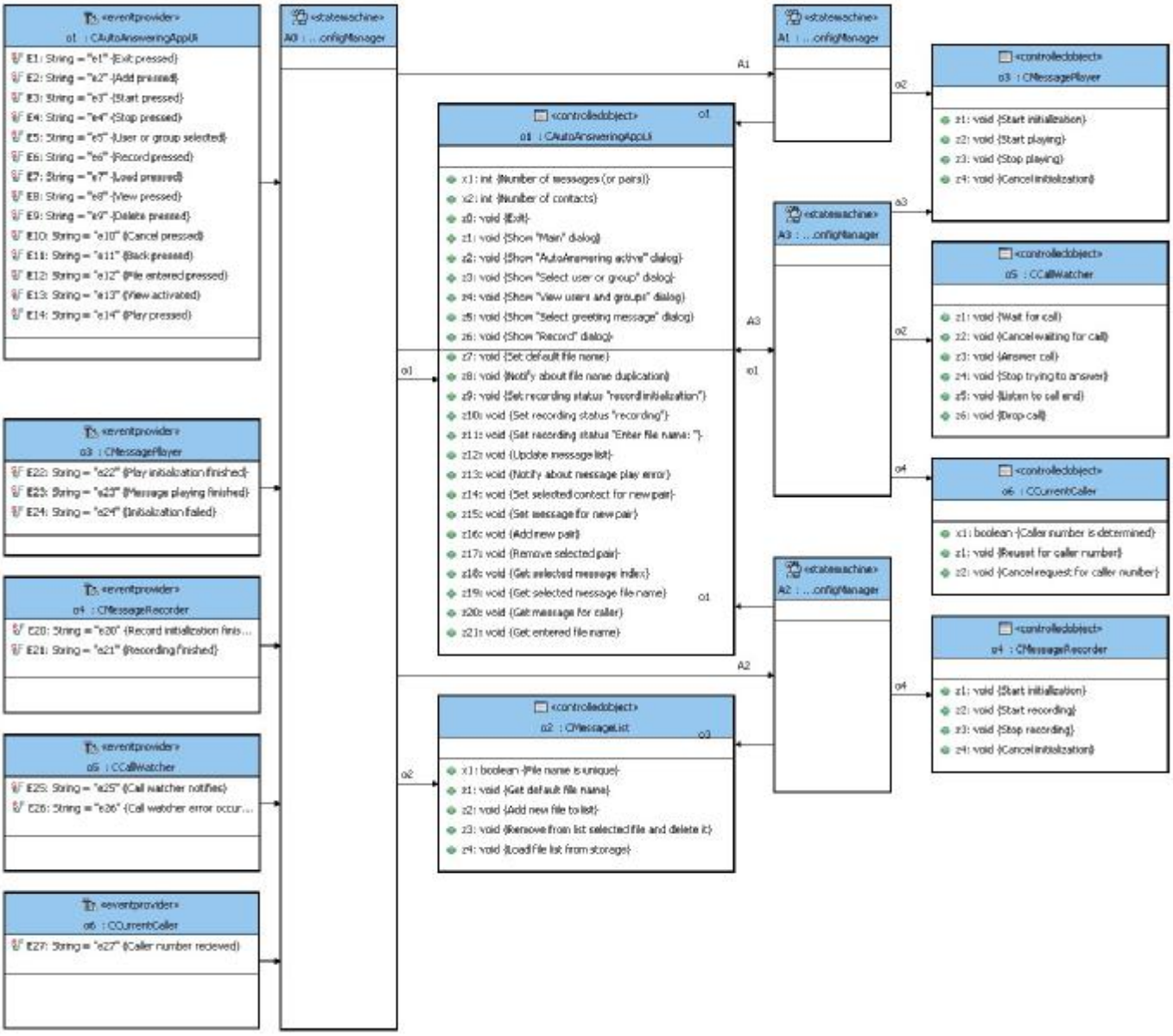

Рис. 8. Схема связей автоматов

Класс статической модели *AutoAnsweringAppUi* в динамической модели декомпозирован на четыре автомата:

- *A0*. Построен на основе диаграммы переходов между экранными формами (рис. 6). Управляет режимом настройки голосовых сообщений.
- *A1*. Управляет проигрыванием сообщений.
- *A2*. Контролирует запись новых сообщений.
- *A3*. Управляет режимом ответов на звонки.

На рис. 9–12 показаны диаграммы переходов автоматов *A0–A3*.

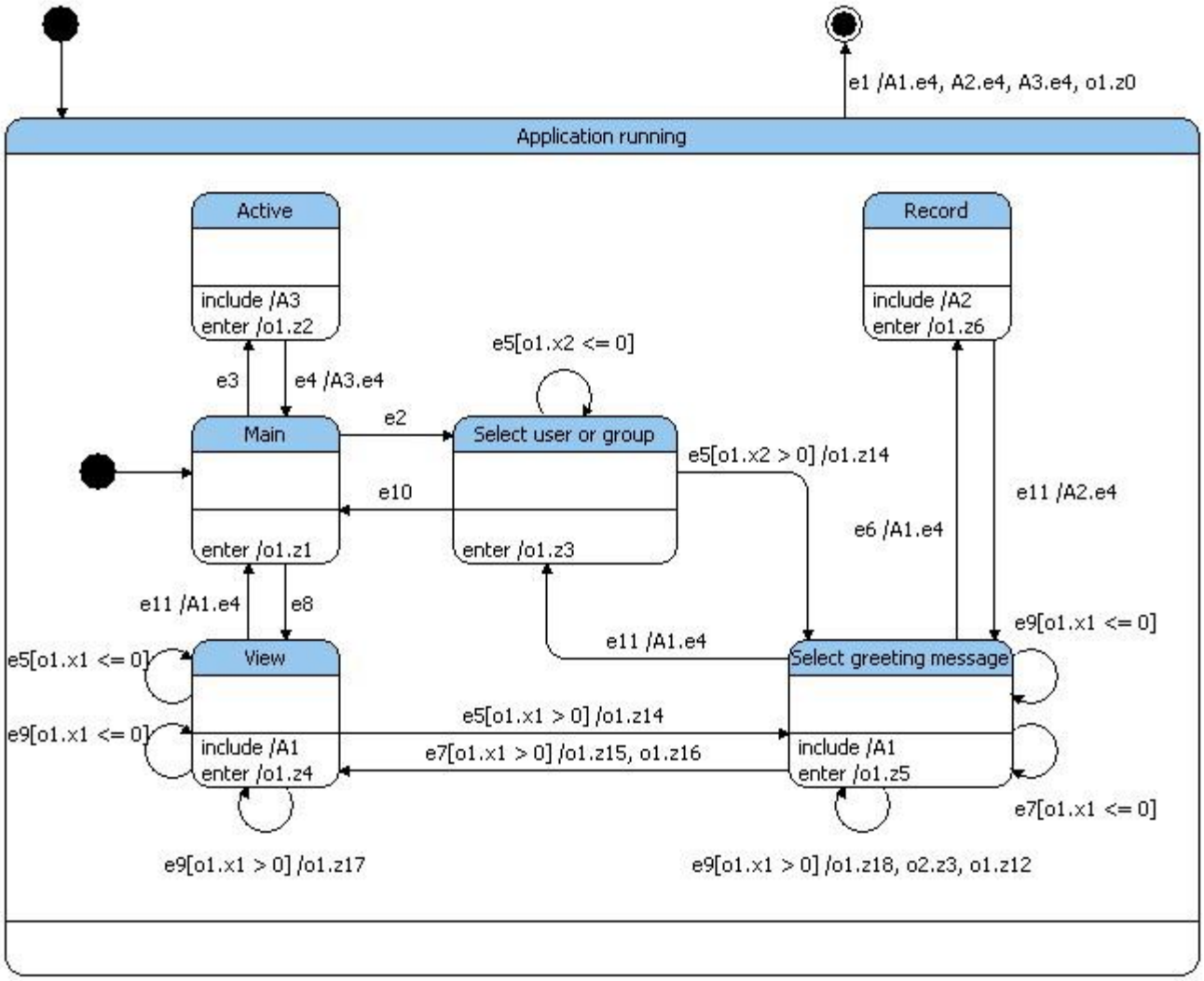

Рис. 9. Диаграмма переходов автомата *A0*

Автомат *A0* взаимодействует с автоматами *A1–A3* по вложенности (например, автомат *A3* вложен в состояние *Active* автомата *A0*) и вызываемости (например, в автомате *A0* присутствует переход *e11/A2.e4*).

Вложенность означает, что все события, полученные автоматом A0 в состоянии *Active* будут также делегированы автомату *A3*.

Вызываемость означает явную посылку события другому автомату.

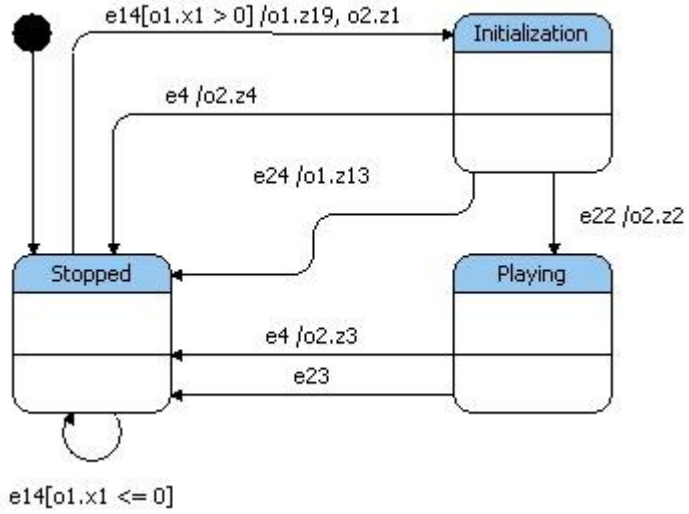

Рис. 10. Диаграмма переходов автомата *A1*

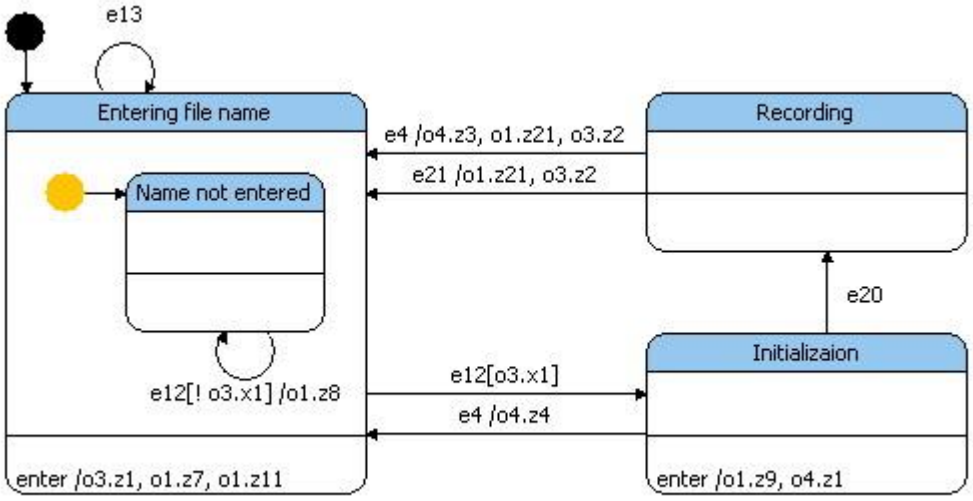

Рис. 11. Диаграмма переходов автомата *A2*

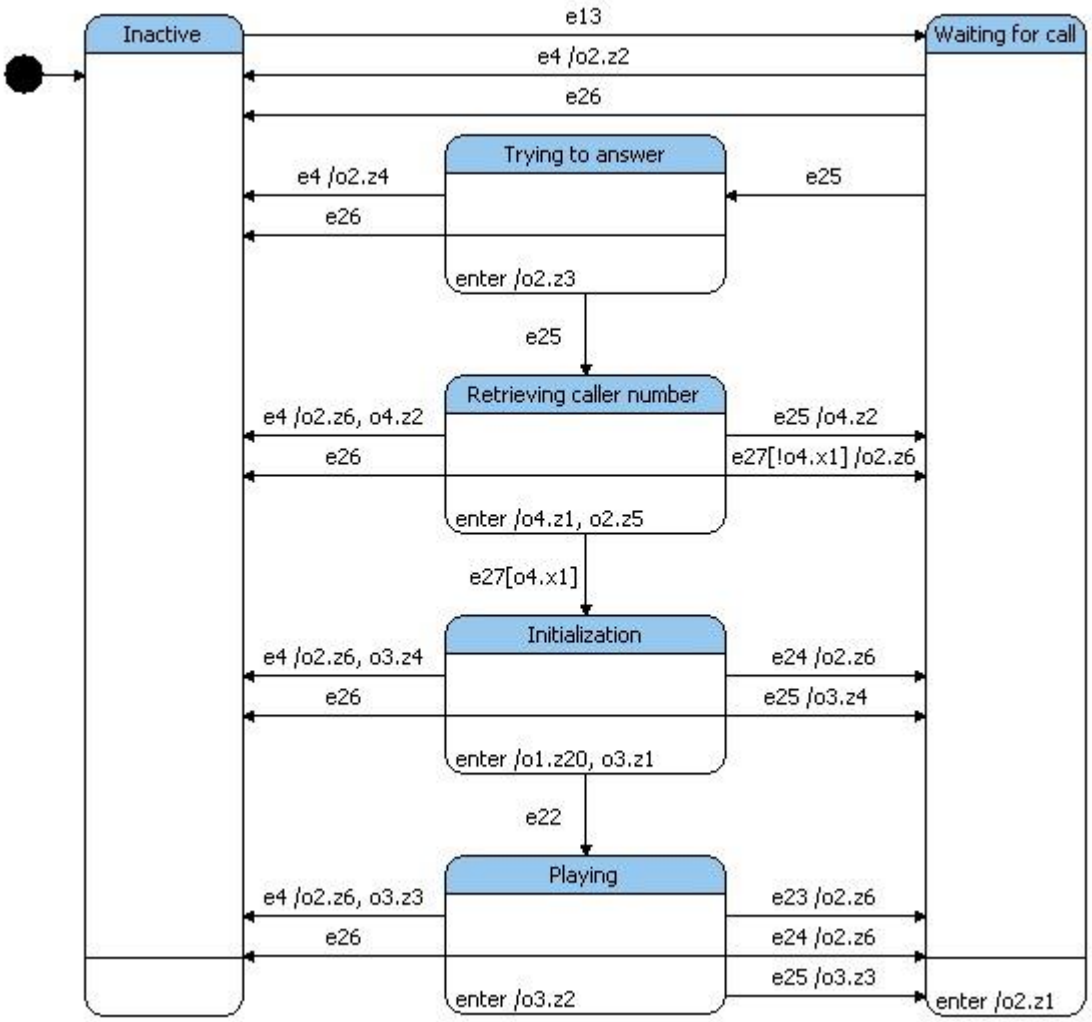

Рис. 12. Диаграмма переходов автомата *A3*

3.3. Создание кода

Для того чтобы поставщик событий имел возможность посылать события автоматной модели, у него должна быть ссылка на экземпляр «движка» автоматной модели.

Для класса `CMessagePlayer` метод, реализующий установку ссылки на «движок» автоматной модели, выглядит следующим образом:

```
void CMessagePlayer::SetModelEngine(CAutoAnswererModelEngine* aModelEngine)
    {
    iModelEngine = aModelEngine;
    }
```

Для посылки события автомату, поставщик событий должен вызвать метод `HandleL` у объекта «движка» автоматной модели. Например, поставщик событий `CMessagePlayer` посылает автомату событие о завершении инициализации следующим образом:

```
void CMessagePlayer::MapcInitComplete(TInt aError, const TTimeIntervalMicroSeconds& /*aDuration*/)
    {
    iLogger.LogVariableL("MessagePlayer.Error", aError);
    if (iModelEngine)
        {
```

```
        if (aError == KErrNone)
            {
            TRAPD (err, iModelEngine->HandleL(CAutoAnswererModelEngine::E23));
            if (err)
                {
                Stop();
                }
            }
        else
            {
            iStatus = aError;
            iModelEngine->HandleL(CAutoAnswererModelEngine::E24);
            }
        }
    }
```

Для объектов управления методы, реализующие входные и выходные переменные, должны быть запрограммированы вручную. Так для объекта управления `CMessageList` выходная переменная `z3` имеет вид:

```
void CMessageList::Z3L(TStateMachineContext& aContext)
    {
    if (aContext.iMessageFileIndex >= 0 && aContext.iMessageFileIndex < iList->Count())
        {
        TBuf<256> fileName;
        StringLoader::Load(fileName, R_AUTOANSWERING_MSG_DIR_PATH);
        fileName.Append(iList->At(aContext.iMessageFileIndex).iName);
        BaflUtils::DeleteFile(iFs, fileName);
        iList->Delete(aContext.iMessageFileIndex);
        CreateDefaultNameL();
        }
    }
```

Входная переменная `x1` для того же объекта управления реализуется следующим образом:

```
TBool CMessageList::X1L(TStateMachineContext& aContext)
    {
    TInt numberMessages(iList->Count());

    for (TInt i = 0; i < numberMessages; i++)
        {
        const TMessage& message = iList->At(i);

        if(message.iName.Compare(*aContext.iFileName) == 0)
            {
            return EFalse;
            }
        }
    return ETrue;
    }
```

Приведем код, сгенерированный для метода `TransiteOnEventL` автомата *A2*, реализующий переходы из текущего активного состояния, каждый из которых зависит от события и значений входных переменных:

```
TA2::EState TA2::TransiteOnEventL(CAutoAnswererModelEngine::EEvent aEvent, TStateMachineContext& aContext)
    {
    TA2::EState targetState = UNKNOWN_STATE;
    for (TA2::EState sourceState = iState; targetState == UNKNOWN_STATE && sourceState != TOP; sourceState =
GetSuperstate(sourceState))
        {
        switch (sourceState)
            {
            /* State: Top */
            case TOP:
                switch (aEvent)
                    {
                    default: ;// Do nothing if no event was triggered
                    }
                break;
            /* State: Recording */
            case RECORDING:
                switch (aEvent)
                    {
                    case CAutoAnswererModelEngine::EEvent::E21:
```

```
                    iO1->Z21L(aContext);
                    iO3->Z2L(aContext);
                    targetState = ENTERING_FILE_NAME;
                    EnterStateL(targetState, aContext);
                    break;
                case CAutoAnswererModelEngine::EEvent::E4:
                    iO4->Z3L(aContext);
                    iO1->Z21L(aContext);
                    iO3->Z2L(aContext);
                    targetState = ENTERING_FILE_NAME;
                    EnterStateL(targetState, aContext);
                    break;
                default: ;// Do nothing if no event was triggered
                }
            break;
        /* State: Initializaion */
        case INITIALIZAION:
            switch (aEvent)
                {
                case CAutoAnswererModelEngine::EEvent::E20:
                    targetState = RECORDING;
                    EnterStateL(targetState, aContext);
                    break;
                case CAutoAnswererModelEngine::EEvent::E4:
                    iO4->Z4L(aContext);
                    targetState = ENTERING_FILE_NAME;
                    EnterStateL(targetState, aContext);
                    break;
                default: ;// Do nothing if no event was triggered
                }
            break;
        /* State: Entering file name */
        case ENTERING_FILE_NAME:
            switch (aEvent)
                {
                case CAutoAnswererModelEngine::EEvent::E12:
                    if (iO3->X1L(aContext))
                        {
                        targetState = INITIALIZAION;
                        EnterStateL(targetState, aContext);
                        }
                    break;
                case CAutoAnswererModelEngine::EEvent::E13:
                    targetState = ENTERING_FILE_NAME;
                    EnterStateL(targetState, aContext);
                    break;
                default: ;// Do nothing if no event was triggered
                }
            break;
        /* State: Name not entered */
        case NAME_NOT_ENTERED:
            switch (aEvent)
                {
                case CAutoAnswererModelEngine::EEvent::E12:
                    if (!iO3->X1L(aContext))
                        {
                        iO1->Z8L(aContext);
                        targetState = NAME_NOT_ENTERED;
                        EnterStateL(targetState, aContext);
                        }
                    break;
                default: ;// Do nothing if no event was triggered
                }
            break;
        }
    }

    // Look for default transition
    for (TA2::EState sourceState = iState; targetState == UNKNOWN_STATE && sourceState != TOP; sourceState =
GetSuperstate(sourceState))
    {
    switch (sourceState)
        {
        /* State: Top */
        case TOP:
            break;
        /* State: Recording */
        case RECORDING:
            break;
        /* State: Initializaion */
        case INITIALIZAION:
            break;
        /* State: Entering file name */
```

```
        case ENTERING_FILE_NAME:
            break;
        /* State: Name not entered */
        case NAME_NOT_ENTERED:
            break;
        }
    }

// If no transition was found stay in current state
return targetState != UNKNOWN_STATE ? targetState : iState;
}
```

Для инициализации и запуска автоматной модели необходимо создать объекты управления, поставщики событий и автомат:

```
iMessageList = CMessageList::NewL(Document().Fs());

iContactEngine = CPbkContactEngine::NewL();

iActiveView = CAutoAnsweringViewActive::NewL();
AddViewL(iActiveView); // Transfer ownership to base class

iMainView = CAutoAnsweringViewMain::NewL();
AddViewL(iMainView); // Transfer ownership to base class

iRecordView = CAutoAnsweringViewRecord::NewL();
AddViewL(iRecordView); // Transfer ownership to base class

iSelectMsgView = CAutoAnsweringViewSelectMsg::NewL(*iMessageList);
AddViewL(iSelectMsgView); // Transfer ownership to base class

iSelectUserView = CAutoAnsweringViewSelectUser::NewL(*iContactEngine);
AddViewL(iSelectUserView); // Transfer ownership to base class

iViewView = CAutoAnsweringViewView::NewL(Document().ContactMessageMap(), *iContactEngine);
AddViewL(iViewView); // Transfer ownership to base class

iCallWatcher = CCallWatcher::NewL(*iLogger);
iMessageRecorder = CMessageRecorder::NewL(Document().Fs());
iMessagePlayer = CMessagePlayer::NewL(Document().Fs(), iCallWatcher->Phone(), *iLogger);
iMessagePlayer->SetUseLoudSpeaker(ETrue);
iLineMessagePlayer = CMessagePlayer::NewL(Document().Fs(), iCallWatcher->Phone(), *iLogger);
iLineMessagePlayer->SetUseLoudSpeaker(EFalse);
iCurrentCaller = CCurrentCaller::NewL(*iContactEngine);

iModelEngine = CAutoAnswererModelEngine::NewL(
    this, iMessageList, iMessagePlayer, iMessageRecorder,
    iCallWatcher, iCurrentCaller);
iMessagePlayer->SetModelEngine(iModelEngine);
iMessageRecorder->SetModelEngine(iModelEngine);
iCallWatcher->SetModelEngine(iModelEngine);
// ToDo find out what should do LineMessagePlayer
iLineMessagePlayer->SetModelEngine(iModelEngine);
iCurrentCaller->SetModelEngine(iModelEngine);
```

4. ЗАКЛЮЧЕНИЕ

Предлагаемая методика создания приложений для мобильных устройств базируется на понятии «состояние» и позволяет описывать динамическую модель системы в виде множества взаимодействующих конечных автоматов, задаваемых в форме графов переходов.

Подход, предлагаемый в настоящей работе, позволяет использовать автоматы при спецификации, в тексте программы и при протоколировании.

Разработан метод формального и изоморфного перехода от спецификации к текстам программ на языке *C++*. Формальность перехода позволяет для одного графа переходов использовать его в качестве полного теста для сертификации построенной программы, а изоморфность с графом переходов позволяет проверять программу сверкой ее текста.

Конечные автоматы все шире начинают использоваться при проектировании программного обеспечения мобильных устройств, что в частности изложено в книге [12], одна из начальных глав которой называется «Наш друг конечный автомат».

5. ЛИТЕРАТУРА

1. *Грехем И.* Объектно-ориентированные методы. Принципы и практика. М.: Вильямс, 2004. – 880 с.
2. *OMG Model Driven Architecture*. http://www.omg.org/mda/
3. *Буч Г., Рамбо Г., Якобсон И.* UML. Руководство пользователя. М.: ДМК, 2000. – 358 с.
4. *Mellor S., Balcer M.* Executable UML: A Foundation for Model Driven Architecture. MA: Addison-Wesley, 2002. – p. 258.
5. *Шалыто А. А.* SWITCH-технология. Алгоритмизация программирования задач логического управления. СПб.: Наука, 1998. – 628 с. http://is.ifmo.ru/books/switch/1
6. *Шалыто А.А., Туккель Н.И.* SWITCH-технология — автоматный подход к созданию программного обеспечения "реактивных" систем //Программирование. 2001. № 5, с. 45–62. http://is.ifmo.ru/works/switch/1/
7. *Шалыто А.А., Туккель Н.И.* Танки и автоматы //BYTE/Россия. 2003. № 2, с. 69–73. http://is.ifmo.ru/works/tanks_new/.
8. *Harrison R.* Symbian OS C++ for Mobile Phones. John Wiley & Sons, 2003. – p. 826.
9. *Гуров В.С., Нарвский А.С., Шалыто А.А.* Исполняемый UML из России // PC Week/RE. 2005. № 26, с.18,19. http://is.ifmo.ru/belletristic/triedin/
10. *Гуров В.С., Мазин М.А., Нарвский А.С., Шалыто А.А.* UML. SWITCH-технология. ECLIPSE //Информационно-управляющие системы. 2004. № 6, с.12–17. http://is.ifmo.ru/works/uml-switch-eclipse/
11. *http*://unimod.sourceforge.net
12. *Салпре И.* Программирование мобильных устройств на платформе .Net Compact Framework. М.: Вильямс. 2006. – 550 с. http://is.ifmo.ru/progeny/mobdev/

Лист регистрации изменений

Номера листов (страниц)					*Всего листов (страниц) в докум.*	*№ документа*	*Входящий № сопроводительного докум. и дата*	*Подп.*	*Дата*
Изм.	*Измененных*	*Замененных*	*Новых*	*Аннулированных*					