Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Гудиев Артур Владимирович

Графический DSL для разработки мобильных приложений

Выпускная квалификационная работа

Научный руководитель:
д.ф.-м.н., проф. А.Н. Терехов

Рецензент:
ООО "НМТ", ген. директор,
В.В. Оносовский

Санкт-Петербург

2019

SAINT-PETERSBURG STATE UNIVERSITY
Chair of Software Engineering

Artur Gudiev

Graphic DSL for mobile development

Graduation thesis

Scientific supervisor:
Prof. Andrey Terekhov

Reviewer:
NMT Ltd, CEO,
Valentin Onosovski

Saint-Petersburg

2019

Оглавление

Введение

На сегодняшний день для разработки мобильных приложений используется большое количество платформ, языков и методов. При этом существующие инструменты довольно разрозненны [11], и задача выработки общей технологии по-прежнему актуальна.

Существуют различные способы высокоуровневого описания мобильных приложений, например, архитектурные паттерны mvc, pac, microkernel и пр. [8] [9] [19]. Однако они были заимствованы из других сфер разработки ПО и не вполне соответствуют специфике мобильных приложений [13]. Настольные и веб-приложения зачастую используются для длительных сессий, предоставляют пользователю информационную среду с широким исследовательским простором. Мобильные же приложения обычно используются для более коротких сессий, более сфокусированы на выполнении конкретных задач [7] [3].

Использование подходящего архитектурного паттерна позволяет значительно увеличить эффективность разработки приложения, но еще большего результата можно достигнуть благодаря использованию графических языков. Графическое представление приложений удобно для понимания архитектуры разрабатываемого ПО, позволяет избежать многих ошибок еще на этапе проектирования. Стандартом в области графических языков общего назначения является UML [24]. Однако UML не позволяют максимально эффективно описывать специфику конкретных типов приложений, так как стандарт довольно большой по объему и все-таки не является универсальным [32]. Для описания приложений конкретной предметной области используются графические DSL (Domain Specific Languages) [21]. DSL — язык программирования в терминах конкретной предметной области, применяющийся для решения узкого круга задач. Графические DSL-языки помогают представлять приложения с помощью визуальных

диаграмм, по которым затем может быть сгенерирован финальный код приложения.

На кафедре системного программирования СПбГУ много лет проводятся исследования в сфере графических DSL. В частонсти, были разработаны визуальные среды моделирования QReal и REAL.NET, позволяющие создавать графические DSL [29] [34]. Кроме того, реализованы некоторые конкретные графические DSL: QReal:Robots [33], предназначенный для программирования роботов, Real-IT [37], прозволяющий разрабатывать телекоммуникационные системы. Была также создана методология разработки DSL [32].

Существенной частью DSL является целевая мобильная платформа, для которой будет генерироваться финальный код приложения. UbiqMobile — платформа для разработки кроссплатформенных мобильных приложений [27]. В приложениях UbiqMobile бизнес-логика исполняется на сервере, соединение с которым поддерживает тонкий клиент на устройстве пользователя. На кафедре системного программирования СПбГУ проводились различные исследования, связанные с платформой Ubiq Mobile [36] [39].

Целью данной работы является разработка архитектурного шаблона логической структуры мобильного приложения и создания на его основе графического DSL, позволяющего описывать основную логическую структуру приложения в терминах контроллеров, состояний и условий переходов между ними. По описанию приложения должен генерироваться целевой программный код. Графический DSL должен быть частью фреймворка, содержащего библиотеку шаблонных конструкций языка и позволяющую расширять её для нужд разных классов мобильных приложений.

Для достижения этой цели в рамках работы были сформулированы следующие задачи.

- Разработка архитектурного шаблона логической структуры мобильного приложения.
- Разработка графического DSL на основе разработанного шаблона.
- Добавление возможности генерации кода для платформы UbiqMobile.
- Апробация DSL на демонстрационных примерах.

1 Обзор

Графические DSL активно применяются в сфере мобильной разработки, и их использование имеет ряд особенностей. Так, DSL может надстраиваться над конкретной мобильной платформой (App Inventor [10]). В этом случае приложение описывается в терминах компонент пользовательского интерфейса конкретной платформы. По описанию генерируется код главной экранной формы приложения, а также некоторые обработчики событий. Однако основная логика приложения реализуется вручную.

Надстройка также возможна и над различными сторонними библиотеками (для игр, конкретных бизнес-систем и т.д.) (GameSalad [4]). Приложение описывается в терминах узкой предметной области. К примеру, для игр это — действия, роли и т.д. Кроме того, целевой платформой может служить среда кросс-платформенной мобильной разработки (Xamarin [25]).

1.1 Концепция графических DSL

Визуальное моделирование стало применяться при разработке ПО во второй половине прошлого века. Визуальное моделирование облегчает поддержку концептуальной целостности больших проектов [28]. Кроме того, оно выполняет коммуникативную роль для многочисленных участников проекта [32]. Язык визуального моделирования (графический язык) представляет из себя набор графических компонент и правил составления графических моделей.

Можно выделить следующие уровни абстракции визуального моделирования: предметная область, модель и метамодель. Под предметной областью в сфере информационных технологий подразумевается та сфера, для которой предназначено разрабатываемое ПО. Для облегчения определенной работы создается упрощеное описание предметной области — модель. Под метамоделью

понимается язык, с помощью которого можно создатваь всевозможные модели данной предметной области.

Предметно-ориентированные языки (Domain-Specific Languages, DSL) — языки, предназначенные для решения задач конкретной предметной области. DSL-языки помогают эффективно использовать абстракции, благодаря чему достигается оптимальность в реализации ПО при относительно небольшой трудоемкости создания самого DSL. DSL позволяют ускорить разработку ПО в несколько раз [16].

1.1.1 Real-IT

В 1997-2002 гг. на кафедре системного программирования СПбГУ совместно с ГП «Терком» и ЗАО «ЛанитТерком» был реализован проект Real [17] [23]. Данный проект был посвящен разработке универсального инструмента моделирования и предметно-ориентированного решения для разработки телекоммуникационных систем путем переноса существующего решения RTST [38] на Real. Другой проект Real-IT [6] [5] является предметно-ориентированным решением для разработки информационных систем, интенсивно работающих с данными. Real-IT разрабатывался на основе Real.

Технология Real-IT предназначена для быстрой разработки (моделирования и автоматической генерации) приложений, бизнес-логика которых целиком определяется схемой данных. Модель данных в Real-IT задаётся с помощью диаграмм классов UML, ограничения на эту модель определяются с помощью диаграмм объектов, на основе специального языка ограничений [31]. Модель пользовательского интерфейса строится с помощью специальных программных инструментов. Real-IT автоматически генерирует диаграммы классов UML, описывающие созданную модель интерфейса. С помощью Real-IT было создано более десяти промышленных систем [32].

1.1.2 QReal

На кафедре системного программирования СПбГУ была разработана среда QReal [29]. QReal — это кроссплатформенный свободно распространяемый инструмент с открытым исходным кодом, предназначенный для создания специализированных сред визуального программирования [15]. Используя набор инструментов платформы QReal, разработчик специализированного графического языка создает ряд формальных описаний этого языка — задает его метамодель (перечисляет набор сущностей языка, возможные связи между ними и их свойства), определяет формы элементов, описывает правила ограничения, налагаемые на них, задает правила преобразования моделей для автоматического получения исходного кода, описывает семантику сущностей языка и т.п.

Решение QReal состоит из следующих частей. Графический интерфейс получаемых сред разработки предоставляет пользователям такие инструменты, как редакторы моделей, диаграмм и свойств элементов, «миникарту», окно для отображения ошибок и предупреждений и пр. Репозиторий представлен в виде объектно-ориентированной базы данных, хранящую все создаваемые в QReal модели. Репозиторий представлен в виде динамически загружаемой библиотеки языка C++, соответственно он может использоваться в сторонних программах через специальный интерфейс. Для обеспечения многопользовательской работы есть возможность версионирования создаваемых моделей с помощью систем контроля версий. Кроме того, разработаны отладчики и интерпретаторы на уровне моделей, позволяющие проследить исполнение алгоритма и проверить его корректность. Генераторы кода позволяют автоматически получить программный или тестирующий код. Механизм проверки правил ограничений проверяет корректность заданных моделей. Он может включаться при изменении свойства элемента, перемещении, создании и удалении объекта и пр. Для

определенного рода изменений таких, как групповое изменение имени, инвертирование связей, выделение набора элементов в подпрограмму и пр., определен механизм рефакторинга моделей, позволяющий внести изменение во внутреннюю структуру системы и при этом не изменяющий ее внешнее поведение.

1.1.3 QReal:Robots

QReal:Robots — среда для визуального программирования роботов [30]. QReal:Robots успешно используется для преподавания информатики во многих школах России [35]. Данная среда для визуального программирования роботов отличается от аналогичных решений (например, Robolab [14]) наличием русификации и возможностью отладки программы.

QReal:Robots основан на модели вычислений, ориентированной на поток исполнения [34]. Блок языка соответствует элементарной команде роботу, связи между блоками отражают последовательность, в которой выполняются блоки. Все блоки языка делятся на смысловые группы. Группа «Алгоритмы» включает в себя блоки, отвечающие за последовательность выполнения команд. В группу «Алгоритмы» входят такие блоки, как «Диаграмма поведения робота», «Линия соединения», «Условие» и «Цикл». В группе «Действия» представлены блоки, соответствующие элементарным командам: «Гудок», «Играть звук», «Моторы вперёд», «Моторы назад», «Моторы стоп», «Параллельные задачи» и «Функция». В группе «Инициализация» расположены блоки, обозначающие начало и конец программы и задающие начальное состояние подсистем робота: «Блок инициализации», «Начало», «Конец» и «Сбросить показания энкодера». Группа «Ожидания» содержит блоки, останавливающие выполнение программы до наступления некоторого события: «Ждать свет», «Ждать сенсор касания», «Ждать цвет», «Таймер» и пр. Кроме того, в QReal:Robots есть возможность использования специфических функций операционной системы и сторон-

них библиотек, написанных на текстовых языках.

1.2 Существующие DSL для разработки мобильных приложений

В данном разделе расмотрены два существующих графических DSL-языка для разработки мобильных приложений: Mobl [26] и XIS-Mobile [18]. Для каждого DSL-языка приведены его отличительные особенности. В конце раздела изложены недостатки обоих DSL-языков.

1.2.1 Mobl

Mobl — это текстовый статически типизированный компилируемый язык, представленный в виде плагина для Eclipse [26]. Язык Mobl имеет ряд преимуществ, направленных на повышение производительности мобильных разработчиков.

Пользовательские интерфейсы, определенные с помощью Mobl, задаются декларативным образом и реагируют на изменения в состоянии приложения в отличие от других подходов, где состояние и представление строго разделены.

В Mobl предусмотрены языковые конструкции для определения моделей данных с помощью объявления сущностей. Изменения в объектах сущностей автоматически сохраняются в базе данных. Запросы к данным также выполняются на уровне сущностей и не требуют встроенных SQL-запросов.

Язык Mobl статически типизирован, что позволяет использовать такие функции IDE, как выделение ошибок, разрешение ссылок и автодополнение кода. Тем не менее, из-за логического вывода типов во многих случаях не нужно явно указывать типы данных. В тех случаях, когда динамическая типизация предпочтительнее, можно использовать тип Dynamic, предоставляя произвольный

доступ к свойствам и методам динамических переменных.

Логику приложения можно описать на языке сценариев, который очень похож на типизированную версию JavaScript. Код написан в синхронном стиле и автоматически переводится компилятором в асинхронный код JavaScript с использованием различных преобразований.

1.2.2 XIS-Mobile

XIS-Mobile — фреймворк, основанный на UML и позволяющий описывать структуру мобильных приложений платформо независимым методом [18]. XIS-Mobile поддерживает генерацию кода для Android и Windows Phone. XIS-Mobile составлен из нескольких частей.

Пакет **Entities View** используется для идентификации сущностей, которые имеют отношение к предметной области. Entities View содержит в себе пакеты Domain View и BusinessEntities View. Domain View описывает сущности на основе диаграммы классов, атрибуты сущностей и отношения между ними в терминах ассоциаций, агрегаций и наследования. BusinessEntities View отображает объекты более выского уровня для предоставления определенного контекста и пространства взаимодействия.

Пакет **UseCases View** подробно описывает действия, которые может выполнять актор при взаимодействии с приложением в контексте бизнес сущностей или с помощью внешнего сервиса.

Architectural View отражает взаимодействие между мобильным приложением и другими объектами. Данные взаимодействия представлены в виде ассоциаций, которые подключают мобильное приложение к сервисам.

Пакет **User-Interfaces View** состоит из NavigationSpace и InteractionSpace пакетов, которые используются для описания UI форм.

1.2.3 Выводы

Недостатком описанных DSL решений является отсутствие четкой архитектурной модели представления мобильного приложения. Данные DSL не позволяют описывать логическую архитектуру приложения, представлять ее в виде отдельных логических модулей, связывающих логику приложения, сложные потоки данных и экранные формы в единое целое.

1.3 Инструменты создания DSL

Далее будут рассмотрены два фреймворки для создания графических DSL-языков: MetaEdit+, Modelling SDK и QReal. Для каждого из них приводится краткое описание его характеристик.

1.3.1 MetaEdit+

MetaEdit+ — это среда для создания DSL. MetaEdit+ разрабатывается компанией MetaCase при участии университета города Ювяскюля, Финляндия (University of Jyvaskyla). MetaEdit+ позволяет определить предметно-ориентированный язык, содержащий сущности предметной области и различные правила метамодели, а также генератор объектного кода. MetaEdit+ предоставляет средства для редактирования диаграмм, генерации документации и поддержки различных платформ. Кроме того, в MetaEdit+ есть возможность многопользовательского режима работы, объединения нескольких предметно-ориентированных языков, отладки генераторов, автоматического перехода от сгенерированного кода к соответствующей части модели по клику. Репозиторий MetaEdit позволяет производить автоматическое обновление DSL-языка для всех пользователей и их моделей.

1.3.2 Modelling SDK

Для генерации графического DSL языка используется технология Modelling SDK for Visual Studio. Modelling SDK — это технология, предназначенная для разработки визуальных DSL. Modeling SDK позволяет интегрировать в Visual Studio собственные визуальные предметно-ориентированные языки программирования [12].

Разработка визуального предметно-ориентированного языка происходит следующим образом (см. рис. 1). Создается проект разработки DSL, затем разрабатывается и редактируется метамодель (описывается множество всех синтаксически корректных диаграмм), генерируются реализованные классы, затем происходит компиляция и отладка DSL-пакета в новом экземпляре Visual Studio.

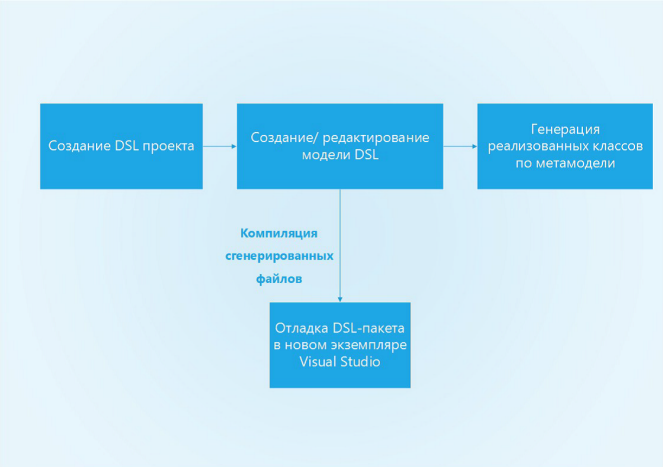

Рис. 1: Процесс разработки DSL

Для программирования метамодели используется графический редактор Modelling SDK, но также можно переопределить или добавить новые методы в сгенерированные частичные классы языка C#.

Для генерации используется язык описания шаблонов T4 [22], представляющий из себя совокупность управляющих команд на языках C# или Visual

Basic. Для каждого нового DSL решения автоматически создаются проекты Dsl и DslPackage. В Dsl хранятся различные артефакты метамодели создаваемого DSL, а в DslPackage — настройки пользовательского интерфейса целевого графического редактора.

Для хранения моделей используются доменные классы — абстракции сущностей хранимых данных в модели. Так же есть классы фигур, свойства которых связаны со свойствами классов моделей посредством механизма событий в языке C# [2]. В SDK имеются способы представления и хранения, а также валидации моделей. Любой доменный класс можно пометить атрибутом, который наделяет его возможностью быть валидированным. [42]

1.3.3 QReal

QReal — это кроссплатформенный свободно распространяемый инструмент с открытым исходным кодом, предназначенный для создания специализированных сред визуального программирования [15]. Архитектура QReal основывается на шаблоне проектирования Model/View. Метамодель разрабатываемого языка описывается в виде XMLформата довольно простой структуры. Графические изображения элементов задаются с помощью текстового языка SDF, являющегося расширением языка описания векторной графики SVG [29] [34]. На кафедре системного программирования СПбГУ активно ведутся исследования в области QReal [41] [40].

1.3.4 Выводы

Данные инструменты предоставляют удобные средства для разработки DSL-языков. Целевой платформой разрабатываемого DSL будет платформа для создания мобильных приложений UbiqMobile. Ввиду того что UbiqMobile разра-

ботан на основе платформы .NET, основным инструментом создания DSL была выбрана технология Modelling SDK от компании Microsoft.

1.4 Фреймворки для разработки мобильных приложений

В данном разделе приводится описание трех фреймворков для создания мобильных приложений: Xamarin, Android Studio и Ubiq Mobile. Каждый из перечисленных фреймворков может служить целевой платформой DSL-языка для автоматической генерации кода конечного мобильного приложения.

1.4.1 Xamarin

Технология Xamarin предназначена для кроссплатформенной разработки мобильных приложений [25]. Данный фреймворк предоставляет основной язык разработки — C#, библиотеку классов и среду выполнения, которые работают на платформах iOS и Android. Примерно 90% кода приложений Xamarin является кроссплатформенным. Xamarin может компилировать довольно производительные нативные приложения. Xamarin содержит основные компоненты, соответствующие элементам Android SDK и iOS SDK. Данные компоненты строго типизированы, что позволяет осуществлять проверку типов во время компиляции и разработки.

В Xamarin предусмотрена возможность вызова библиотек Objective-C, Java, C и C++, что упрощает использование существующего кода, созданного для целевых платформ. Кроме того, фреймворк предоставляет современные языковые методы, такие как, параллельное программирование, функциональные конструкции и динамический доступ к элементам.

1.4.2 Android Studio

Android Studio является официальным IDE для разработки мобильных приложений для Android [1]. Android Studio основана на среде разработки Intelij IDEA. В дополнение к мощным инструментам IntelliJ, Android Studio предлагает дополнительные средства, увеличивающие производительность разработчика, такие как гибкая основанная на Gradle система сборки, скоростной эмулятор, обширные инструменты тестирования, поддержка C++ и Google Cloud Platform.

1.4.3 UbiqMobile

UbiqMobile — это фреймворк для разработки мобильных интерфейсов к информационным системам для бизнеса [27]. Данная платформа предоставляет возможность разрабтывать кроссплатформенные мобильные приложения, в которых бизнес-логика исполняется на сервере, соединение с которым поддерживает тонкий клиент на устройстве пользователя. На текущий момент приложения UbiqMobile используются в сфере медицины, логистики и клининга. Сама технология представлена в виде плагина к Visual Studio.

Основная идея платформы состоит в терминальной архитектуре, когда все приложения исполняются на сервере, а мобильное устройство рассматривается как удаленный терминал. Обмен данными между клиентом и сервером осуществляется через проприетарный двоичный протокол, построенный над TCP/IP. Система воспринимает различные действия пользователя как некоторые события, которые обрабатываются с помощью API платформы. Платформа UbiqMobile предоставляет некоторые API для доступа как к внешнему миру, так и к разлиным частям функиональности системы. **GrpahicsAPI** предоставляет базовое API для обмена информацией с мобильным клиентом. **MessagingAPI** —

внутреннее API, обеспечивающее основные функции синхронного и асинхронного взаимодействия между приложениями через сообщения и почтовые ящики. **GMapAPI** предназначено для создания мобильных версий сервисов, использующих картографическую информацию сервиса Google Maps. **SNetAPI** — это API для доступа к различным социальным сетям.

На кафедре системного программирования СПбГУ велись различные исследования, связанные с платформой Ubiq Mobile [36] [39].

1.4.4 Выводы

Рассмотренные инструменты предоставляют удобные средства для создания мобильных приложений. Важной особенностью фреймоворков Xamarin и UbiqMobile является их кроссплатформенность. Разработка DSL-языка с генератором кода для фреймворка, поддерживающего кроссплатформенность, позволит создавать по визуальным диаграммам мобильные приложения, которые будут работать на основных мобильных платформах. Технология UbiqMobile, частично разрабатывавшаяся на кафедре системного программирования СПбГУ была выбрана в качестве целевой платформы DSL-языка для разработки мобильных приложений.

2 Архитектурный шаблон мобильного приложения

Исходя из специфики мобильных приложений, удобным способом описания и реализации их логической структуры является использование диаграмм переходов состояний (State transition diagram) [20]. Для этих целей был разработан шаблон контроллеров и состояний, позволяющий управлять состояниями, переходами между ними, группировать их и переиспользовать.

Контроллеру приложения соответствует некоторый целостный фрагмент логики. У контроллера имеются различные состояния, каждому из которых может соответствоать своя экранная форма. Помимо экранной формы в состоянии описана простая алгоритмическая структура передачи данных между экранными формами, условными блоками, некоторыми фрагментами кода и вызовами сервисов. В каждый момент времени у контроллера активно лишь одно состояние. Состояния могут переключаться, а также передавать различные данные. Результатом смены состояния является передача фрагмента данных, логически законченного и понятного другому состоянию.

Состояния и условия переходов удобно группировать в контроллеры по их логической связности, по общности данных и форм, с которыми они работают, по частоте переходов и передачи данных между состояниями. Группировка состояний в контроллеры позволяет более жестко определить логику переходов, разрешая переходы между состояниями внутри контроллера и запрещая их между состояниями несвязанных контроллеров.

Главный цикл приложения управляется специальным механизмом шины, запускающей и переключающей контроллеры состояний. У каждого контроллера есть точка входа и возможность задавать входные параметры при переключении на него. Выходов из контроллера может быть несколько. Возможны такие

варианты выхода, как «вернуться к вызвавшему контроллеру», «переключиться на заданный контроллер» и т.д. Внутри контроллера логика выполнения реализуется в виде конечного автомата. У каждого контроллера есть набор предопределенных состояний (в частности, начальное и конечное состояния), и при программировании контроллера можно добавлять новые состояния.

Реализация мобильного приложения с помощью модели контроллеров состояний позволяет централизовать его логическое ядро, структура кода становится наглядной и ее легко охватить одним взглядом. При использовании контроллеров задача разработчика мобильного приложения сводится к тому, чтобы описать нужные логические контроллеры, состояния и условия переходов.

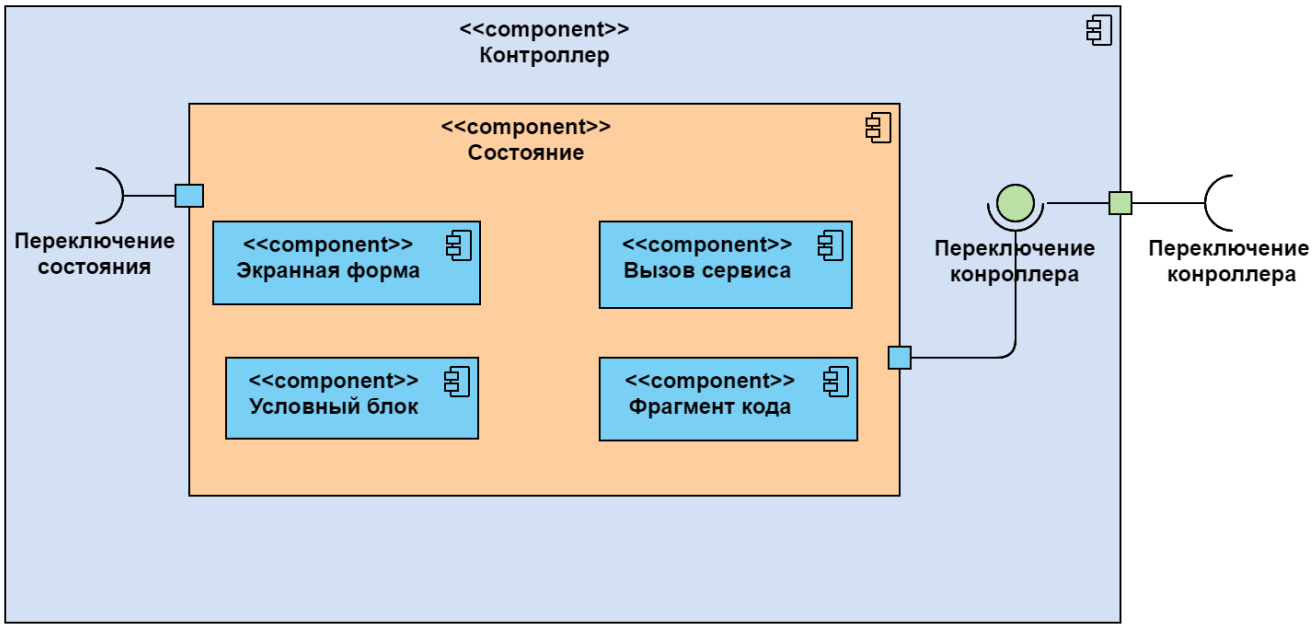

Рис. 2: Компоненты архитектурного шаблона

3 DSL-язык для разработки мобильных приложений

В данной главе представлено описание копонент DSL-языка, детали его реализации в Modelling SDK, а также пример программы, которую можно составить с помощью данного DSL.

3.1 Метамодель языка

Модель контроллеров состояний была апробирована на мобильных приложениях разных классов и доказала свою эффективность. Но еще лучших результатов можно достигнуть, взяв эту модель за основу графического DSL для мобильных приложений. DSL состоит из универсальных, легко параметризируемых, настраиваемых и переиспользуемых элементов.

Логика мобильного приложения подразделяется на логические компоненты - контроллеры. Контроллеры (Controller) могут иметь несколько состояний (State). В процессе работы контроллер меняет свое текущее состояние, а после определенного события может передать управление другому контроллеру. Контроллеры и состояния содержат в себе порты. Порты могут быть входными (InPort) и выходными (OutPort). С помощью связи для передачи данных (DataLink) можно посылать различные наборы данных между контроллерами и состояниями. Наборы данных указываются в параметре Data связи DataLink.

Внутри каждого состояния исполнение логики начинается с входного порта. Затем последовательно исполняются блоки кода, соответствующие элементам, с которыми связан входной порт, после чего исполняются блоки, следующие за данными элементами, и т.д. Среди таких элемент может быть UI форма (ShowForm). В параметрах у нее указаны графические примитивы и события, по

которым затем происходит передача управления. Кроме того, подобным элементом может быть условный блок (ConditionalBlock), вызов сервиса (ServiceCall) и определенный блок кода, полностью задаваемый пользователем (AppCode).

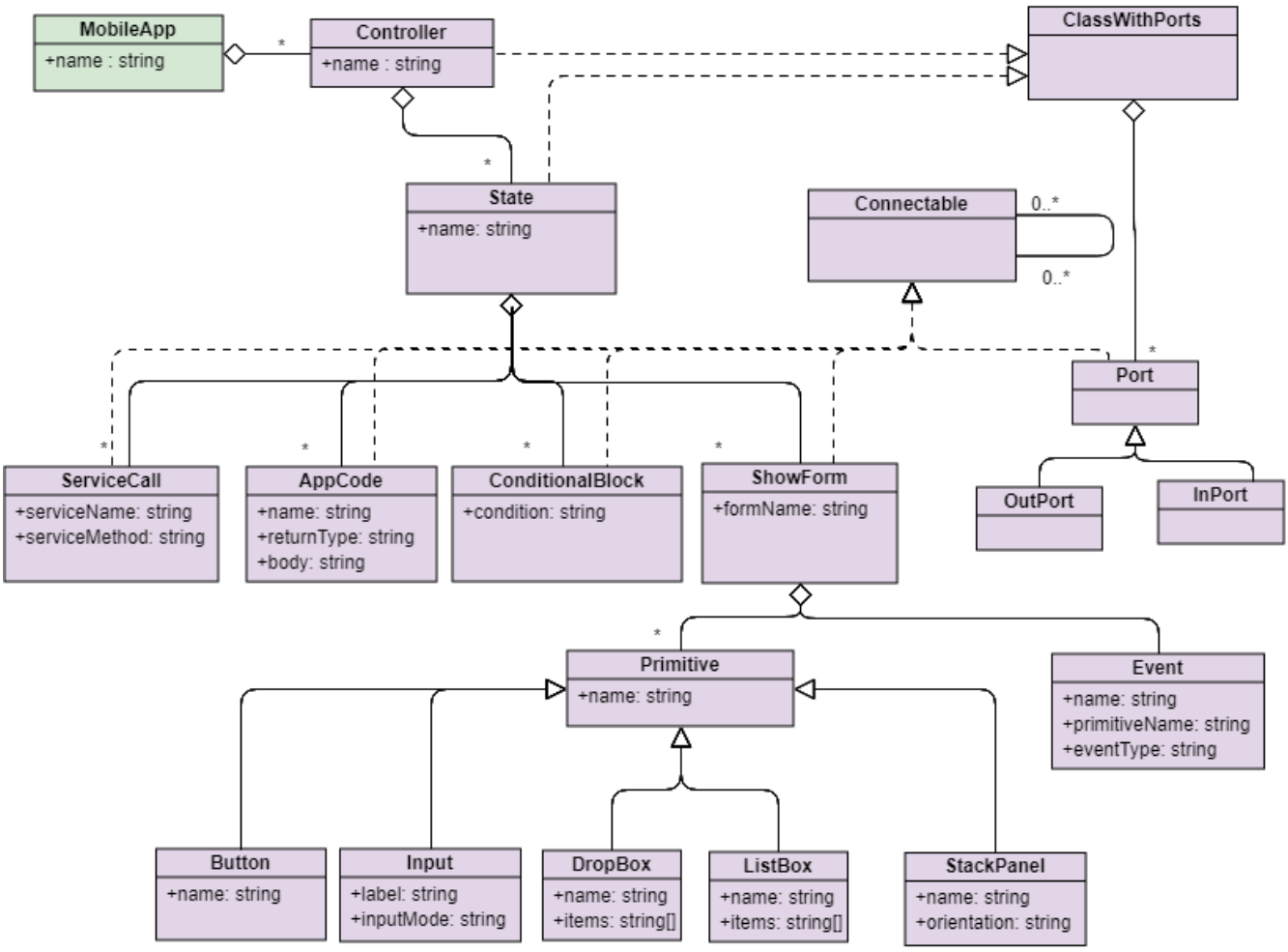

Рис. 3: Метамодель разработанного DSL

3.2 Описание языка

Язык описания мобильных приложений реализован с помощью технологии Modelling SDK. Главными компонентами языка являются доменные классы и связи между ними. Доменные классы содержат различные свойства, которые

можно отображать на диаграмме с помощью соответствующих декораторов. Свойства задаются при создании объектов доменного класса. Связи бывают нескольких типов: связь встраивания и ссылочная связь. Для того чтобы можно было создавать новые объекьы на диаграмме доменные классы связываются также и с геометрическими формами, которые отвечают за внешний вид объектов на диаграмме.

В данном языке есть следующие доменные классы: Controller, State, Service, DataLink, ShowForm, ServiceInvoke, ServiceMsg. У всех классов есть различные поля, например, поле имени. Также разработан внутренний класс для формы — GraphicElement. Это базовый класс для графических примитивов, которые будут отображаться в пользовательском интерфейсе. У многих примитивов есть дополнительные поля, необходимые для корректной работы, например, текст, и внутренние элементы.

Классы соединяются между собой различными связями. Например, класс Controller ссылается на класс State. Для создания связи на диаграмме необходимо создать соответствующий коннектор.

3.2.1 Controller, State, DataLink

Основными элементами языка являются контроллер (см. рис. 2) и его состояния. Состояния размещаются на контроллере, могут соединяться между собой, а также с портами своего контроллера для описания условий входа и выхода из него. Каждый контроллер имеет уникальное имя, начальное состояние, соединяется с другими контроллерами через входные и выходные порты.

Каждое состояние раскрывается в отдельную диаграмму, на которой описываются условия входа, выхода из состояния и его внутренняя логика. В логику состояний входит отображение экранных форм, обработка их событий, обраще-

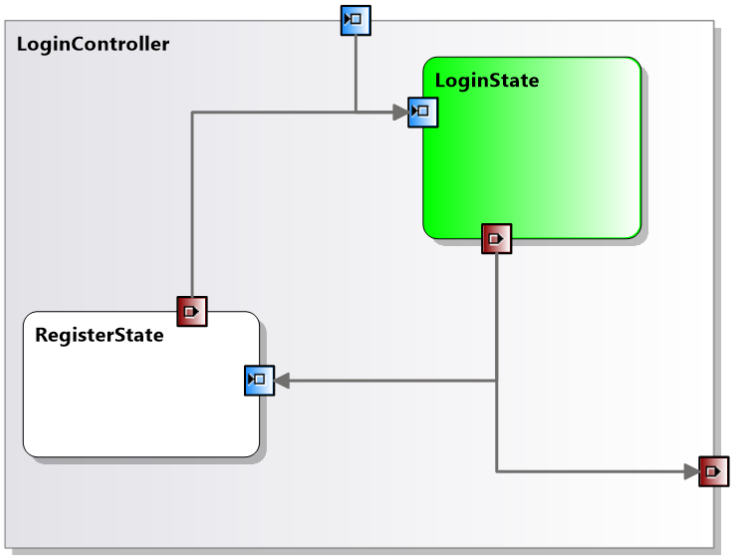

Рис. 4: Controller

ние к сервисам, источникам данных, проверка условий и т.д.

Между контроллерами, состояниями и другими элементами языка могут передаваться данные, для этого используется элемент-связь DataLink (см. рис. 3). В качестве передаваемых данных могут выступать: данные экранной формы; результат выполнения пользовательского или сервисного метода; данные, полученные от сервиса; результат проверки условий и т.д.

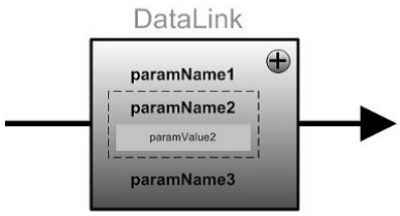

Рис. 5: DataLink

3.2.2 ShowForm, UIDataLink

Часто при переходе в состояние происходит отображение экранной формы, а события её элементов могут обрабатываться в этом состоянии обращениями к сервисам, базе, или же сменой состояния. Для того, чтобы связать существующую экранную форму с состоянием, либо описать новую, используется элемент ShowForm(см. рис. 4). В связке с элементом-связью DataLink он позволяет удобно описывать данные, передаваемые из формы к другим элементам.

Для гибкой настройки параметров отображения формы элемент используется вместе с элементом-связью UIDataLink (см. рис. 5). С его помощью легко связать данные, передаваемые на вход формы с её элементами, описать какие данные, где должны быть отображены, какие элементы нужно скрыть и т.д.

Рис. 6: ShowForm

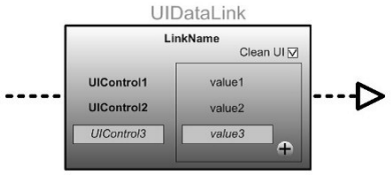

Рис. 7: UIDataLink

3.2.3 ServiceInvoke, ServiceMsg

Часто мобильные приложения имеют сервис-ориентированную архитектуру. В языке поддержана возможность обращения к сервисам с помощью элементов ServiceInvoke (см. рис. 6) и ServiceMsg (см. рис. 7). Элемент ServiceInvoke позволяет описывать вызовы сервисных методов. Основные параметры элемента:

имя сервиса и имя метода. Для обработки асинхронных сообщений от сервиса используется элемент ServiceMsg. Его параметры: имя сервиса, тип входящего сообщения и тип данных, переданных в этом сообщении.

Рис. 8: ServiceInvoke

Рис. 9: ServiceMsg

3.3 Пример

В качестве примера использования языка выбрано приложение шаблона пошаговой игры (см. рис. 9). Между контроллерами и состояниями определены переходы, в которых передаются некоторые фрагменты данных: из LoginController в UserListController передается myUserName – имя авторизованного пользователя как результат работы LoginController. По этому имени в состоянии UserList у удаленного сервиса запрашивается список активных пользователей и выводится на форму UserListForm. В этом списке пользователь может выбрать, с кем начать игру, и тогда данные выбранного пользователя будут переданы как объект UserInfo в состояние InviteUser для создания приглашения. Находясь в состоянии UserList, другой пользователь получает приглашение с его описанием в виде объекта InvitationInfo — инициируется переход в состояние New Invitation для обработки приглашения.

Когда пользователь ввёл свои данные и нажимает кнопку «Войти», срабатывает событие LoginBtnClicked, по которому происходит вызов метода ValidateData

и передача в него данных экранной формы LoginForm. Результат проверки валидации данных передается дальше как параметр DataValid по связи DataLink в элемент If. В случае успешной валидации, происходит переход к вызову метода Login сервиса AuthenticationService. Результат выполнения метода Login также проверяется с помощью If элемента, и в случае успеха происходит выход из состояния Login с передачей в качестве параметра по связи DataLink имени авторизовавшегося пользователя — userName. Если пользователя не удалось авторизовать в системе, то происходит переход назад к элементу ShowForm для формы LoginForm с целью её перерисовки и отображения сообщения об ошибке.

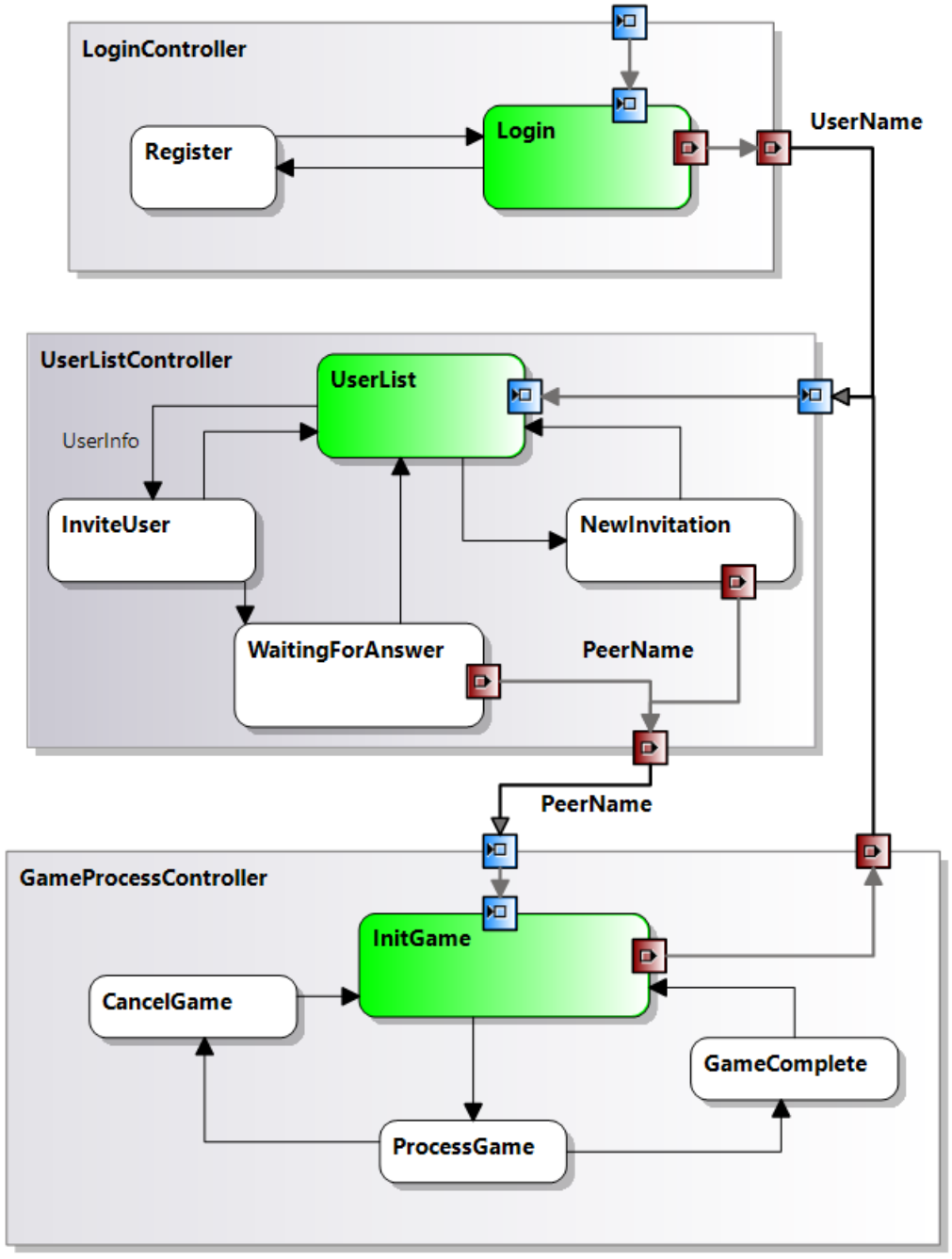

Рис. 10: Шаблон приложения на уровне контроллеров и состояний

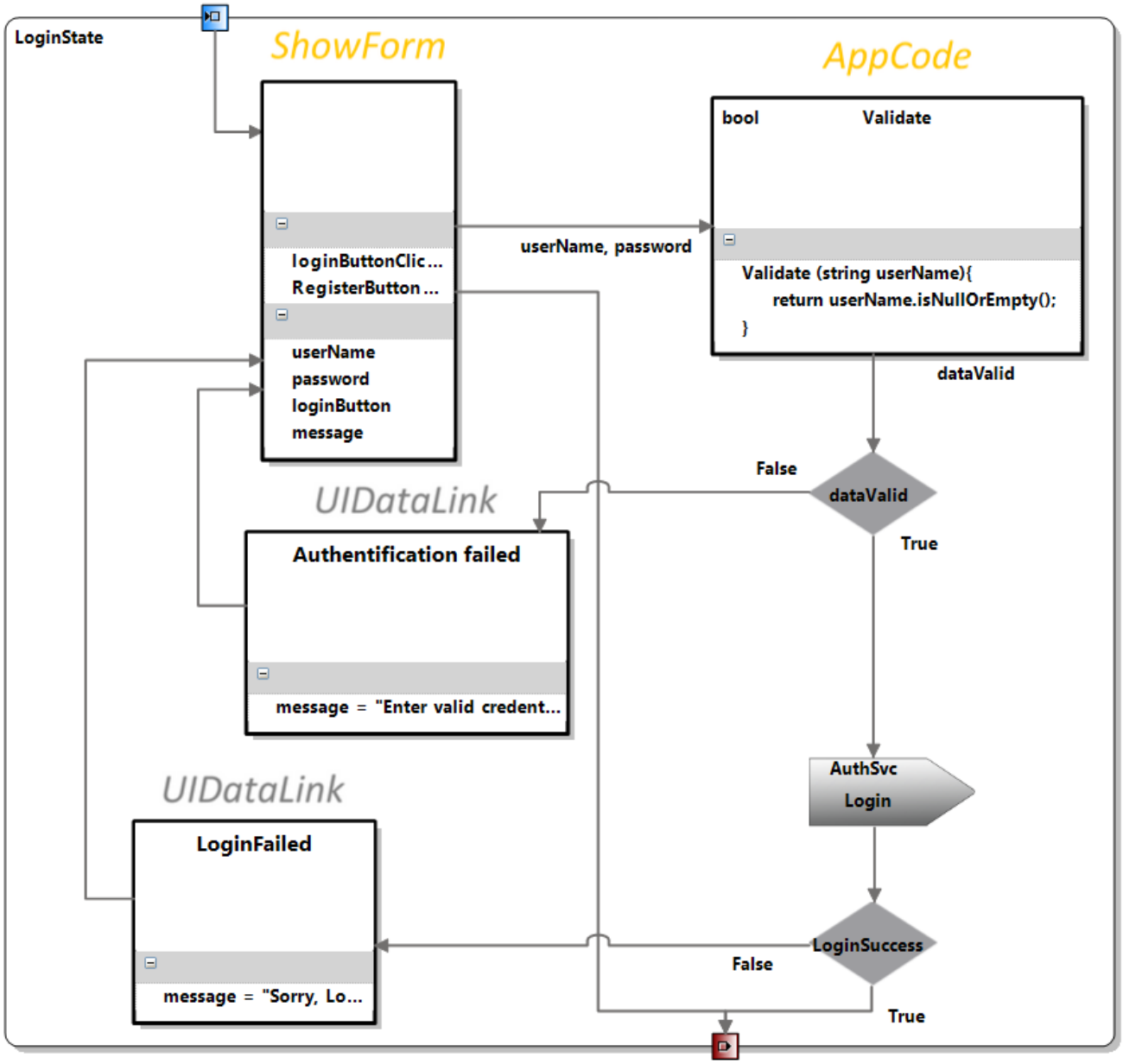

Рис. 11: Содержимое состояния LoginState

4 Кодогенерация

В данной главе представлено описание генератора кода для платформы Ubiq Mobile.

Для генерации кода используется язык текста шаблонов T4. Основными компонентами языка являются директивы, блоки текста и блоки управления. Для генерации неизменяемого кода используются блоки текста, а динамические части реализуются с помощью блоков управления. Целевой платформой является платформа UbiqMobile.

В результате генерации получаются классы контроллеров, которые наследуются абстрактного класса Controller (см. Листинг 1). У каждого контроллера есть несколько состояний, представленных в виде перечислительного типа. Элементы перечислительного типа соответствуют именам состояний. У каждого контроллера есть текущее состояние controllerState, изначально оно инициализировано начальным состоянием, которое вычисляется по диаграмме с помощью функции getFirstState, параметром которой является контроллер.

Процесс работы контроллера реализован в виде конечного автомата. У каждого контроллера есть функция action, которая проверяет, какое состояние является текущим для данного контроллера и вызывает соответствующую функция runState. Например, если у контроллера есть состояния InitialState и GetCodeState, то по ним автоматически будут сгенерированы функции runInitialState и runGetCodeState. По связям, заданным на диаграмме, реализуются шаблоны переходов в контроллерах. Контроллеры могут содержать порты. Они используются для переходов между контроллерами и состояниями.

```
<# foreach (Controller controller in this.ComponentModel.Controllers) { #>

        public class <#= controller.Name #>  : Controller{
```

```
        enum <#= controller.Name #>State {
      <# foreach (State state in controller.States) { #>
            <#= state.Name #>,
      <#}#>
    }

  private <#=controller.Name#>State controllerState =
            <#=controller.Name#>State.<#getFirstState(controller);#>;

        public override void action(){
            <#compileActionFunction(controller);#>
        }

       <# foreach (State state in controller.States) {
            foreach (ShowForm form in state.ShowForms) {
                        compileShowFormFunction(form);
        }
        compileRunState(state);
    }
        }#>
}
```

Листинг 1: Генерация контроллеров и состояний

По коду из Листинга 1 может быть сгенерирован следующий код.

```
        public class MainController : Controller {

        enum MainControllerState {
                ChooseStationState,
                ShowScheduleState,
        }
        private MainControllerState controllerState =
```

```
    MainControllerState.ChooseStationState;

    public override void action () {
            switch (controllerState) {

                    case MainControllerState.ChooseStationState:
                            runChooseStationState ();
                            break;

                    case MainControllerState.ShowScheduleState:
                            runShowScheduleState ();
                            break;
            }
    }

private void runChooseStationState () {...}

private void runShowScheduleState () {...}

}
```

Листинг 2: Фрагмент финального кода

У каждого состояния есть форма отображения. К каждой форме прикреплены графические примитивы. По списку примитивов в конечном коде вызываются соответствующие конструкторы и происходит отображение пользовательского интерфейса.

5 Мобильные приложения, реализованные с помощью DSL

В данном разделе приведено описание мобильных приложений, разработанных при помощи разработанного DSL. Для каждого приложения представлены его логическая структура в терминах компонент языка, а также UI формы платформы Ubiq Mobile, получающиеся после автоматической генерации кода.

5.1 Расписание электричек

Приложение "Расписание электричек"состоит из одного контроллера и двух состояний StartState и ShowState. UI форма состояния StartState содержит в себе различные графические примитивы, такие как поля для выбора станций отправления и назначения, надписи и кнопка "Показать расписание". При нажатии на кнопку происходит переход в состояние ShowState. В состоянии ShowState непосредственно отображается расписание для выбранных станций. При нажатии на кнопку "Назад"произойдет возврат в состояние StartState.

Состояние StartState начинает свою работу с вызова метода getStations сервиса ScheduleSvc. Результатом вызова является список станций stations. Следующим элементом является UIDataLink, который задает значения для экранной формы ChooseStationsForm. После нажатия кнопки ShowButtonPressed срабатывает обработчик события ShowButtonPressed, который вызывает метод validate сервиса ValidationSvc, результат которого записывается в переменную isValid. Далее если isValid равен false, упрвавление снова передается форме, в которой теперь будет сообщение об ошибке. Если isValid равен true, то произойдет переключение на состояние ShowStationsState.

В состоянии ShowStationsState происходит вызов метода getSchedule сервиса

ScheduleSvc. Результат будет записан в переменную scheduleList. Затем при помощи элемента UIDataLink станции отобразятся на форме ShowScheduleForm. При нажатии на кнопку BackButton обработчик события сменит текущее состояние контроллера на состояние StartState.

Переключение между состояниями представлено в виде связей между портами состояний. У каждой связи такого рода есть параметр, в котором можно указать имя контроллер или состояния. Тогда при срабатывании события произойдет переключение в соответствующее состояние или в начальное состояние следующего контроллера. В данном примере у связи события ShowButtonClicked указано состояние ShowState, а у связи события BackButtonClicked - состояние StartState.

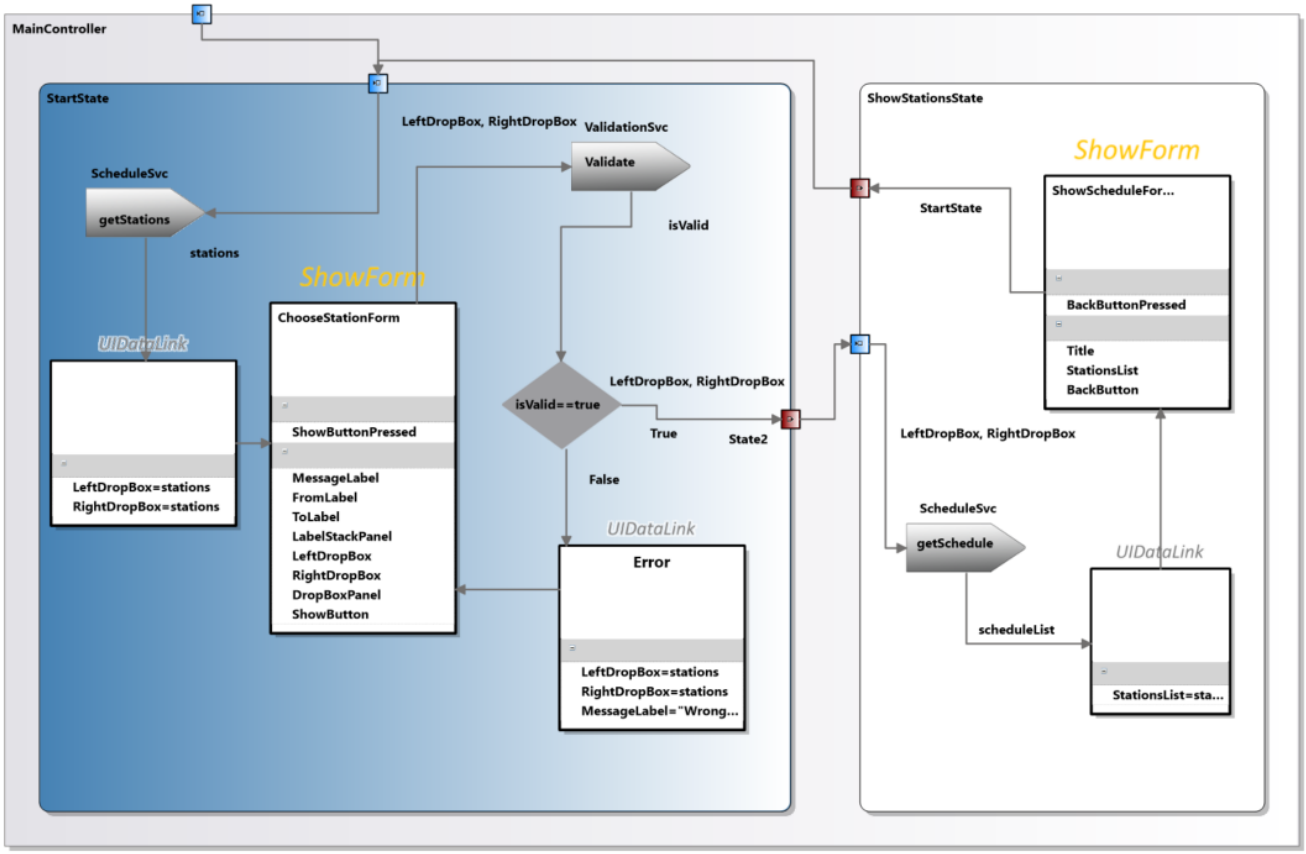

Рис. 12: Схема приложения "Расписание электричек".

UI формы, соответствующие состояниям приложения:

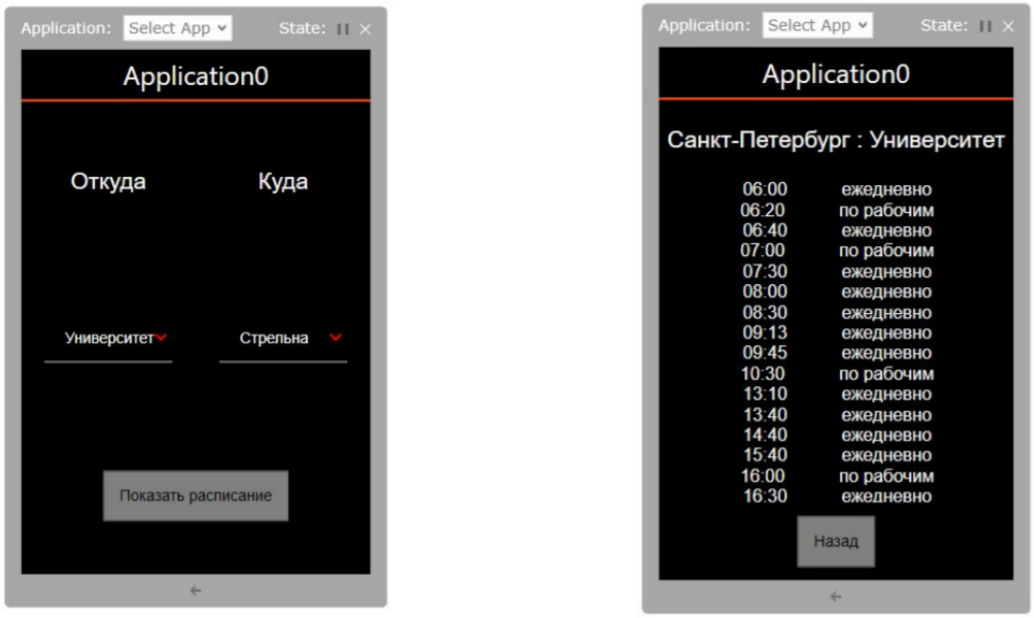

Рис. 13: UI формы "Расписания электричек".

5.2 Приложение с авторизацией

Данное приложение позволяет волонтеры войти в систему. И получить код, который затем может быть использован для получения талонов. В приложении есть два контроллера: LoginController и MainController. Присутствует также переключение между котроллерами, реализованное с помощью портов. В LoginController есть только одно состояние. В MainController присутствуют два состояния: состояние выбора опции и состояние, в котором волонтер может получить необходимый код.

В состоянии LoginState отображается форма LoginForm. В ней обрабатывается событие loginButtonClicked, срабатывающее при нажатии loginButton.

Связь, соответствующая данному событию, соединяет форму с вызовом метода login сервиса LoginSvc. В качестве параметров передаются данные login и password. Результат записывается в переменную loginSuccess. Если loginSuccess равен False, то отобразится форма с сообщением об ошибке. Если же loginSuccess равен true, то произойдет переключение контроллеров. В параметре у связи указано название контроллера MainController. Это означает, что текущий контроллер приложения сменится с LoginController на MainController. После того, как сменится контроллер, активным станет его начальное состояние. У MainController начальное состояние - StartState.

Аналогично обрабатываются события checkInButtonClicked при нажатии на checkInButton и logoutButtonClicked, срабатывающего при клике на кнопку logoutButton. У формы MainMenuForm. В параметрах у связей, соответствующим данным событиям, указаны соответственно состояние CheckInState и контроллер
LoginController. В первом случае произойдет смена состояния у контроллера MainController с StartState на CheckInState. Во втором случае произойдет смена контроллера с MainController на LoginController.

В состоянии StartState контроллера MainController представлена лишь форма, по событиям которой происходит переключение либо на состояние CheckinState, либо на контроллер LoginController. В состоянии CheckinState вызывается метод getKey сервиса CheckinService. Затем ключ передается форме, для отображения на экране.

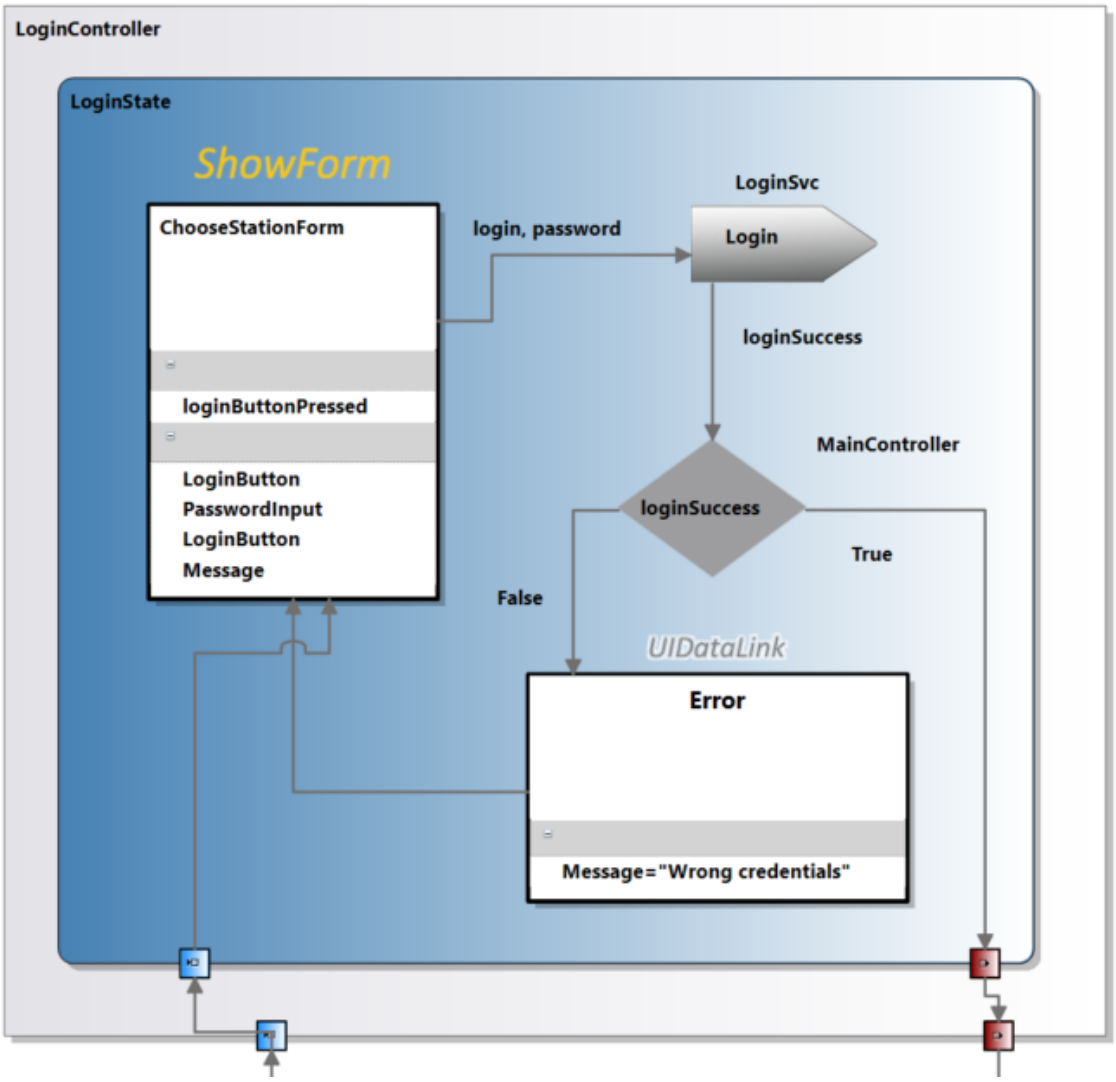

Рис. 14: LoginController

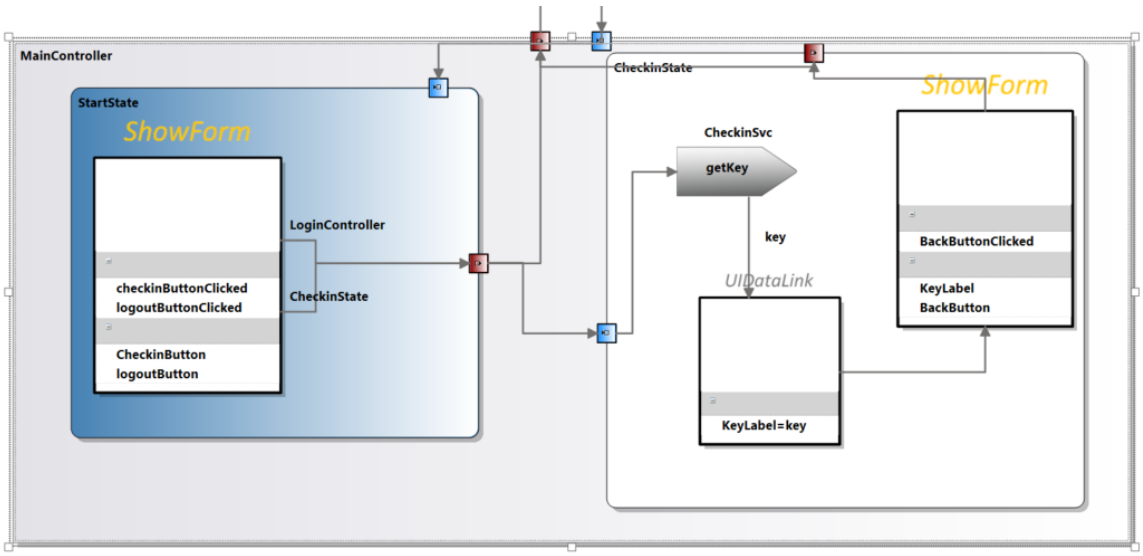

Рис. 15: MainController

После генерации получается следующее приложение:

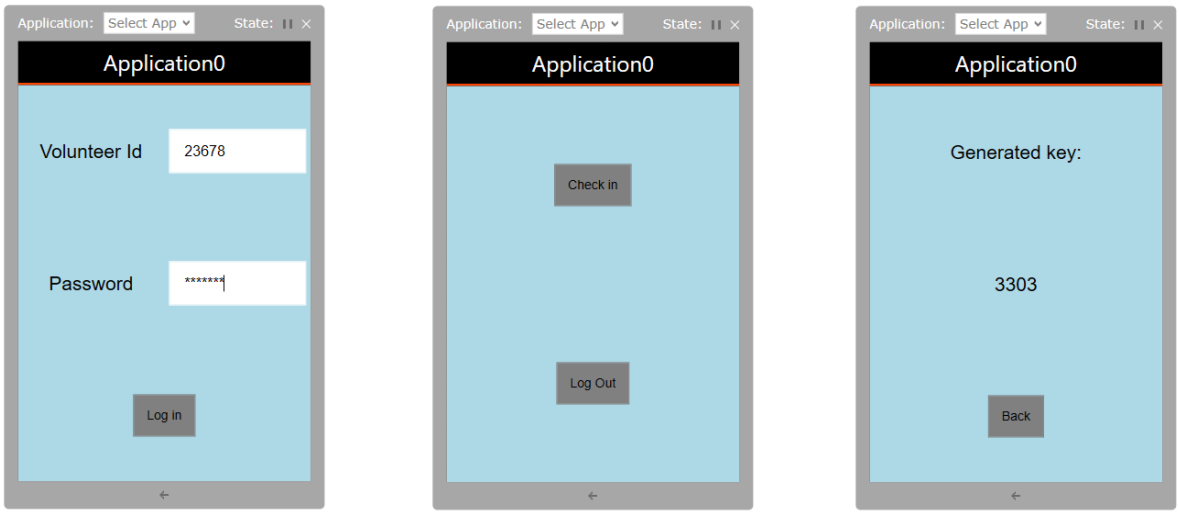

Рис. 16: Экранные формы приложения с авторизацией

Заключение

В рамках данной выпускной квалификационной работы были достигнуты следующие результаты.

- Разработан архитектурный шаблон мобильного приложения с использованием модели контроллеров и состояний.
- На основе шаблона разработан графический DSL для мобильной разработки, состоящий из следующих компонент: контроллеры, порты, состояния, UI формы, сервисы и связи-ссылки для передачи данных.
- Добавлена возможность автоматической генерации кода для целевой платформы UbiqMobile по приложению, описанному средствами DSL.
- Реализованы следующие демонстрационные примеры: расписание электричек, приложение с авторизацией.

Список литературы

[1] Android Studio. — URL: https://developer.android.com/docs (online; accessed: 06.05.2019).

[2] C#. — URL: https://docs.microsoft.com/en-us/dotnet/csharp/ (online; accessed: 25.12.2018).

[3] Flora Harleen, Wang Xiaofeng, Chande Swati. An Investigation on the Characteristics of Mobile Applications: A Survey Study. — 2014. — Oct. — Vol. 11 of I.J. Information Technology and Computer Science. — P. 21–27.

[4] Game Salad. — URL: https://gamesalad.com/ (online; accessed: 25.12.2018).

[5] Ivanov AN. Graphic language for describing constraints on diagrams of UML classes // Programming and Computer Software. — 2004. — Vol. 30, no. 4. — P. 204–208.

[6] Ivanov Alexander N, Koznov Dmitrij V. REAL-IT: MODEL-BASED INTERFACE DEVELOPMENT ENVIRONMENT. — 2005.

[7] Jobe William. Native Apps vs. Mobile Web Apps // International Journal of Information Management. — 2013. — October. — Vol. abs/1001.3489. — URL: https://online-journals.org/index.php/i-jim/article/download/3226/2840.

[8] Leff Avraham, Rayfield James T. Web-application development using the model/view/controller design pattern // Proceedings fifth ieee international enterprise distributed object computing conference / IEEE. — 2001. — P. 118–127.

[9] Lévy Nicole, Losavio Francisca. Analyzing and comparing architectural styles // Proceedings. SCCC'99 XIX International Conference of the Chilean Computer Science Society / IEEE. — 1999. — P. 87–95.

[10] MIT App Inventor. — 2018. — URL: http://appinventor.mit.edu/explore/ (online; accessed: 08.04.2018).

[11] Mobile App Development Tools: A Detailed Comparison. — URL: https://buildfire.com/mobile-app-development-tools/ (online; accessed: 12.05.2019).

[12] Modelling SDK. — URL: https://docs.microsoft.com/en-us/visualstudio/modeling/modeling-sdk-for-visual-studio-domain-specific-l view=vs-2017 (online; accessed: 25.12.2018).

[13] Plakalovic D., Simic D. Applying MVC and PAC patterns in mobile applications // CoRR. — 2010. — Vol. abs/1001.3489. — 1001.3489.

[14] Portsmore Merredith. ROBOLAB: Intuitive robotic programming software to support life long learning // APPLE Learning Technology Review, Spring/Summer. — 1999.

[15] QReal. — URL: http://qreal.ru/static.php?link=vision (online; accessed: 06.05.2019).

[16] R. Pohjonen S. Kelly. Domain-Specific Modeling. — 2002. — August. — P. 161–186.

[17] RTST++: Methodology and a CASE tool for the development of information systems and software for real-time systems / AN Terekhov, K Yu Romanovskii, DV Koznov et al. // Programming and Computer Software. — 1999. — Vol. 25, no. 5. — P. 276–281.

[18] Ribeiro André, da Silva Alberto Rodrigues. XIS-Mobile: A DSL for Mobile Applications. — 2014. — URL: https://dl.acm.org/citation.cfm?id=2554926.

[19] Richards Mark. Software architecture patterns. — O'Reilly Media, Incorporated, 2015.

[20] State transition diagram. — URL: https://www.sciencedirect.com/topics/mathematics/state-transition-diagram (online; accessed: 26.05.2019).

[21] Steven Kelly Juha-Pekka Tolvanen. Domain Specific Modeling. Enable Full Code Generation. Hitchhiker's Guide to the Galaxy Series. — A Wiley-Interscience Publication, 2008. — ISBN: 9780470036662.

[22] T4. — URL: https://docs.microsoft.com/r/visualstudio/modeling/code-generation-and-t4-text-templates?view=vs-2015 (online; accessed: 25.12.2018).

[23] Terekhov Andrey N, Sokolov VV. Implementation of the conformation of MSC and SDL diagrams in the REAL technology // Programming and Computer Software. — 2007. — Vol. 33, no. 1. — P. 24–33.

[24] UML. — URL: https://www.omg.org/spec/UML/ (online; accessed: 06.05.2019).

[25] Xamarin. — URL: https://visualstudio.microsoft.com/ru/xamarin/ (online; accessed: 25.12.2018).

[26] mobl: a DSL for Mobile Web Development. — URL: https://www.infoq.com/articles/Mobl (online; accessed: 06.05.2019).

[27] А.Н.Терехов В.В. Оносовский. Платформа для разработки мобильных приложений UbiqMobile. — URL: http://www.math.spbu.ru/user/ant/all_articles/076_Terekhov_Onos_PlatformaUbiq.pdf.

[28] Брукс Ф. Мифический человеко-месяц или как создаются программные системы. — СПб.: Символ, 1986.

[29] Брыксин Т.А. Платформа для создания специализированных визуальных сред разработки программного обеспечения: диссертация на соискание степени кандидата технических наук // Санкт-Петербургский Государственный Университет. — 2016. — URL: https://disser.spbu.ru/files/disser2/721/disser/nz5QH29b2r.pdf.

[30] Брыксин ТА, Литвинов ЮВ. Среда визуального программирования роботов QReal: Robots // Материалы международной конференции"Информационные технологии в образовании и науке". Самара. — 2011. — P. 332–334.

[31] Иванов АН. Графический язык описания ограничений на диаграммы классов UML // Программирование. — 2004. — no. 4. — P. 35–41.

[32] Кознов Д. В. Методология и инструментарий предметно-ориентированного моделирования : Диссертация на соискание учёной степени доктора технических наук, СПбГУ. — 2016. — URL: http://www.math.spbu.ru/user/dkoznov/papers/PhD2.pdf.

[33] Литвинов Ю.В. Применение DSM-платформы QREAL при разработке среды программирования роботов QREAL:ROBOTS. — 2012. — P. 161–186.

[34] Литвинов Ю.В. Методы и средства разработки графических предметно-ориентированных языков: диссертация на соискание степени кандида-

та технических наук // Санкт-Петербургский Государственный Университет. — 2016. — URL: https://disser.spbu.ru/files/disser2/727/disser/zzT16hbZCm.pdf.

[35] Литвинов Юрий Викторович. Визуальные средства программирования роботов и их использование в школах // Современные информационные технологии и ИТ-образование. — 2012. — no. 8. — P. 858–868.

[36] Малиновский И. К. Проектирование и реализация расширения графического протокола для платформы UbiqMobile. — 2017. — URL: http://se.math.spbu.ru/SE/diploma/2017/bmo/444-Malinovskii-report.pdf.

[37] Терехов А.Н. RTST-технология программирования встроенных систем реального времени // Записки семинара Кафедры системного программирования "Case-средства RTST++". — 1998. — P. 3–17.

[38] Терехов АН. RTST-технология программирования встроенных систем реального времени // Записки семинара кафедры системного программирования"CASE-средства RTST. — 1998. — no. 1. — P. 3–17.

[39] Титов А. Ю. Разработка и реализация специализированного клиентского протокола для платформы Android в системе Ubiq Mobile. — 2012. — URL: http://se.math.spbu.ru/SE/diploma/2012/s/Titov_diploma.doc.

[40] Тихонова М. В. Генерация кода в режиме ”метамоделирования на лету” в системе QReal. — 2015. — URL: http://se.math.spbu.ru/SE/diploma/2015/bmo/444-Tikhonova-report.pdf.

[41] Храмышкина Ю. С. Реализация системы проверки заданий по визуальному моделированию в QReal. — 2017. — URL: http://se.math.spbu.ru/SE/diploma/2017/bmo/444-Khramyshkina-report.pdf.

[42] Шигаров Н. Среда визуального программирования роботов на .NET. — URL: http://se.math.spbu.ru/SE/diploma/2017/bmo/444-Shigarov-report.pdf (online; accessed: 12.05.2019).