Международный журнал информационных технологий и энергоэффективности

Том 7 Номер 1 (23)

2022

Международный журнал информационных технологий и энергоэффективности
Т.7 №1 (23) 2022 г.

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

СОДЕРЖАНИЕ / CONTENT

ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ

1.	**Лашков А.А., Зернов М.М.** Применение навигационного графа для решения задачи поиска маршрута габаритного объекта по цифровой карте местности. **Lashkov A.A., Zernov M.M.** The use of a navigation graph to solve the problem of finding the route of a dimensional object on a digital terrain map	**3**
2.	**Магомедов О. Р., Чернышёв С. А.** Анализ существующих инструментов для кроссплатформенной разработки приложений **Magomedov O. R., Chernyshev S.A.** Analysis of existing tools for cross-platform application development	**17**
3.	**Балашов О.В., Букачев Д.С**. Искусственный интеллект в автоматизации процессов планирования и оперативного управления **Balashov O.V., Bukachev D.S**. Artificial intelligence in automation of planning and operational management processes	**25**
4.	**Магомедов О. Р., Бодренков Г. А., Чернышёв С. А**. Архитектура кроссплатформенных фреймворков **Magomedov O. R., Bodrenkov G. A., Chernyshev S. A**. Architecture of cross-platform frameworks	**31**

Лашков А.А., Зернов М.М. Применение навигационного графа для решения задачи поиска маршрута габаритного объекта по цифровой карте местности // Международный журнал информационных технологий и энергоэффективности. – 2022. – Т. 7 № 1(23) с. 3-16

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 681.3.019

ПРИМЕНЕНИЕ НАВИГАЦИОННОГО ГРАФА ДЛЯ РЕШЕНИЯ ЗАДАЧИ ПОИСКА МАРШРУТА ГАБАРИТНОГО ОБЪЕКТА ПО ЦИФРОВОЙ КАРТЕ МЕСТНОСТИ

[1]Лашков А.А., [2]Зернов М.М.
[1,2]Филиал ФГБОУ ВО «Национальный исследовательский университет «МЭИ» в г. Смоленске, Россия, (214013, г. Смоленск, Энергетический проезд, 1), e-mail: [1]lahkandr010@mail.ru, [2]zmmioml@yandex.ru

Данная статья посвящена рассмотрению возможных путей решения задачи поиска маршрута площадного объекта с использованием навигационного графа. Дано описание навигационного графа. Выделены этапы и особенности реализации построения навигационного графа. Описаны некоторые методы ускорения построения навигационного графа. Рассмотрены методы поиска кратчайшего пути на графах.

Ключевые слова: навигационный граф, КД-дерево, скелет, диаграмма Вороного.

THE USE OF A NAVIGATION GRAPH TO SOLVE THE PROBLEM OF FINDING THE ROUTE OF A DIMENSIONAL OBJECT ON A DIGITAL TERRAIN MAP

[1]Lashkov A.A., [2]Zernov M.M.
Smolensk Branch of the National Research University "Moscow Power Engineering Institute", Smolensk, Russia (214013, Smolensk, Energetichesky proezd, 1), e-mail: [1]lahkandr010@mail.ru, [2]zmmioml@yandex.ru

This article is devoted to the consideration of possible ways to solve the problem of finding the route of an area object using a navigation graph. A description of the navigation graph is given. The stages and features of the implementation of the construction of the navigation graph are highlighted. Some methods of accelerating the construction of a navigation graph are described. Methods of finding the shortest path on graphs are considered.

Keywords: navigation graph, KD-tree, skeleton, Voronoi diagram.

Введение

Поиск маршрута с учётом пространственных размеров перемещаемого объекта – это нахождение последовательности допустимых положений объекта при перемещении его в пространстве от источника к месту назначения. На практике поиск маршрута даже в простых случаях с относительно небольшим количеством препятствий становится трудноразрешимой задачей, если требуется учесть габариты объекта.

Имеется цифровая карта местности (ЦКМ) [1,2,3], описанная в двухмерном пространстве, на которой расположены препятствия (горы, дома, реки, бездорожье и т.д.) и сам перемещаемый объект. Задача заключается в определении кратчайшего маршрута между двумя заданными точками, по которому может пройти объект. При этом необходимо

учитывать проходимость местности, т.к. объект имеет габариты, то он может просто-напросто не пролезть между окружающими его препятствиями или не пройти по бездорожью. Нахождение маршрута для точечного объекта является уже давно известной и решаемой задачей. Тогда как поиск маршрута объекта с учетом его габаритов до сих пор практически не рассматривался. Анализ существующих методов построения маршрута с использованием карты местности позволяет утверждать, что необходимый математический аппарат для нахождения маршрута площадного объекта (цепочка точек соединенные отрезками) уже имеется и необходимо разработать способ на основе известных методов и алгоритмов. В качестве объекта может выступать техника, группа людей, любой крупногабаритный объект, линейные размеры, которого сопоставимы с расстояниями между местными объектами.

Данная статья посвящена рассмотрению возможных путей решения задачи поиска маршрута площадного объекта с использованием графов, совокупности алгоритмов для построения наиболее подходящего вида графов – навигационного графа и алгоритмов поиска пути в графе.

Навигационный граф

Как правило задачи поиска или нахождения маршрута решаются на основе методов на графах и в основном для точечных объектов, например, в геоинформационных системах и компьютерных играх. В первом примере идет речь о графе дорог [4,5], а во втором используется граф точек изменения маршрута [6,7].

Подход на основе графа дорог не учитывает возможности движения по бездорожью и требует дополнительной адаптации на основе анализа проходимости ЦКМ. Это сближает с подходом принятом в компьютерных играх, где на основе анализа игровой карты строится граф точек изменения маршрута (waypoints).

Тем не менее и граф дорого и граф точек перемены маршрута обладают общим недостатком – отсутствием информации о размерах проходимых областей для их соотнесения с размерами(шириной), перемещаемого объекта. Требуемыми свойствами обладает другая структура данных – навигационный граф [8,9].

Навигационный граф является результатом клеточной декомпозиции рассматриваемой среды (ЦКМ). Декомпозиция не является точной, поскольку вычисление основано на дискретном представлении препятствий. В данный граф входят следующие элементы:

- вершины(ячейки) представляют собой окружности (в 3-мерном случае - цилиндры), внутри которых находится свободное от препятствий пространство;
- ребра, соединяющие вершины, являются отрезками, образующимися точками пересечения вершин(окружностей).

Основное свойство навигационного графа состоит в том, что любая точка, принадлежащая данной ячейке, может достигать любой точки, принадлежащей соседней ячейке, проходя мимо любой точки их пересечения (в зависимости от условий). Таким образом нахождение маршрута площадного объекта сводится к задаче поиска пути в навигационном графе. Для этого его сперва необходимо построить.

Процесс построения навигационного графа имеет следующую структуру:

1. Формирование карты проходимости на основе ЦКМ, выделение свободных областей и препятствий.

2. Формирование карты зазоров или дистанционной карты. Для каждой свободной точки находится расстояние до ближайшего препятствия.

3. Для того чтобы компактно расположить вершины навигационного графа по свободной области нужно чтобы вершины занимали как можно больше этой площади, значит, имели максимально возможный радиус. Для выбора таких окружностей может помочь скелет свободной области ЦКМ.

4. Выбор вершин для графа, на основе занимаемых окружностями площадей, и их соединение.

Вышеперечисленные этапы могут приводить к большим вычислительным затратам, чтобы ускорить процесс расчета можно поделить ЦКМ на прямоугольные участки меньшей площади, обеспечивающие быстрый поиск препятствий, например, организованные в виде KD-дерева.

Предварительная обработка цифровой карты местности

В настоящее время цифровое картографирование широко распространено, и во всех отраслях создается все больше геоданных, которые и заносятся на цифровые карты местности. ЦКМ – представление картографических объектов в форме, позволяющей сохранять, обрабатывать и выводить значения их атрибутов компьютерной системой. ЦКМ несет информацию, которую можно анализировать обрабатывать и преобразовывать в формат, который необходим для решения специализированных задач [3].

Нам необходимо преобразовать ЦКМ в форму удобную для построения навигационного графа, а для этого нужно выделить область, занимаемую препятствиями (бездорожьем) и свободную область. В результате решения, данной задачи формируется матрицы проходимости участков карты, определённых в соответствии с регулярной сеткой. Каждая ячейка матрицы будет содержать коэффициент проходимости соответствующего участка (квадрата), изменяемый от 0(свободен) до 1(непроходим или является точкой препятствия). На основе матрицы проходимости, учитывая выбор граничного значения коэффициента проходимости для нахождения точек препятствий (например, выбрать все точки как препятствия, значения коэффициента в матрицы проходимости которых больше 0.5), строят двухмерную карту.

Двухмерная карта представляет собой растровую решетку. Растровая решетка называется прямоугольная решетка Z^2, состоящая из точек евклидовой плоскости с целочисленными координатами. Решетка образуется из точек пресечения двух взаимно перпендикулярных семейств параллельных прямых. Опишем понятие соседства точек:

- непосредственные соседи – точки, евклидово расстояние между которыми равно 1;
- косвенные соседи - точки, расстояние между которыми равно $\sqrt{2}$.

Соответственно можно выделить на основе соседства в растровой решетке 4-смежность и 8-смежность. 4-смежность – смежность, при которой соседними к данной точке являются только непосредственные соседи, а для 8-смежности еще и косвенные соседи. На основе соседства можно сформировать граф смежности, вершинами которого являются точки растровой решетки, а ребрами соединяются все пары соседних вершин. Множество точек может быть связным, если граф смежности является связным, иначе разделенным.

Карта на растровой решетке задается точками решетки двух цветов. Множеством черных точек растровой решетки заданы препятствия, множеством белых – свободные точки. Препятствия являются дискретными фигурами - максимальное связное множество черных точек в растровой решетку. Граничные точки препятствий определяется по соседству 4-х, если какая-либо из четырех соседних точек является свободной, то данная точка является границей.

KD-дерево

Как уже было сказано ранее для упрощения и ускорения построения навигационного графа можно воспользоваться KD-деревом [10,11].

KD-дерево (K-мерное дерево) - структура данных, позволяющая разбить K-мерное пространство делением этого пространства на меньшие участки. Само деление происходит (K-1)-мерными пространствами, например, для плоскости (K=2) прямой.

Такое разбиение позволит сузить диапазон поиска в K-мерном пространстве. Например, поиск ближайшей границы к точке.

Объектами для деления являются точки препятствий, они помещаются в большой ограничивающий прямоугольник (ограничивающим назовем такой прямоугольник, который описывает исходное множество объектов или сам объект, если строится ограничивающий прямоугольник лишь для него), который равен размеру двухмерной карты. Затем данный прямоугольник делится прямой перпендикулярной к одной из сторон, тем самым мы получаем два новых узла дерева (ограничивающих прямоугольника). Новые узлы делятся аналогично и т.д. Деление продолжается по достижению нужной глубины дерева, либо по специальному критерию, либо по количеству объектов в узле.

Опишем некоторые из способов деления узла:

1) В качестве секущей выбрать середину наибольшей стороны.

2) В качестве секущей выбрать такую, что количество объектов слева и справа примерно будет равной.

3) Использовать чередование сторон, т.е. сначала для деления брать середину, допустим, по оси X, далее в новых узлах брать середину по оси Y и т.д.

4) Использовать SAH (Surface Area Heuristic) – функцию оценки разделения ограничивающего прямоугольника [12].

Значение SAH функции вычисляется следующим образом:

$$f(x) = C_t + C_i * \frac{SA_L * N_L + SA_R * N_R}{SA_{parent}},$$

где C_t и C_i – коэффициенты стоимости;

SA_L и SA_R – площади поверхности левого и правого узлов, образуемые в результате деления родительского узла, данным отрезком;

N_L и N_R – количество элементов в левом и правых узлах, после возможного деления родительского узла, данным отрезком;

SA_{parent} – площадь поверхности родительского узла.

Критерий остановки деления узла:

$f_0 = C_i * N$ – SAH родительского узла;

$f_0 < f(x)$ – если выполнено данное неравенство, прекращаем деление.

Т.е. если стоимость прослеживания дочерних узлов больше стоимости прослеживания родительского узла, то деление прекращается.

Дерево строится от корня. В каждом узле дерева хранятся указатели на левое и правое поддеревья, если таковых у узла нет, то он считается листовым (листом). Каждый узел хранит ограничивающий прямоугольник, который описывает объекты данного узла. В листовых и только в листовых узлах хранятся индексы тех объектов, которые входят в данный узел.

Пример структура KD-дерева, описанной на языке C++ представлен ниже [11]:

```
constexpr int N=10; // количество пространств ключей

struct Item {    // структура элемента
  int key[N];    // массив ключей определяющих элемент
  char *info;    // информация элемента
};

struct Node {    // структура узла дерева
  Item i;        // элемент
  Node *left;    // левое  поддерево
  Node *right;   // правое поддерево
}
```

Структура дерева может меняться в зависимости от деталей реализации алгоритма. Например, в узле может содержаться не один элемент, а массив, что повышает эффективность поиска.

Скелетное представление фигуры

Для анализа фигуры описание границ не всегда является информативной и требуется дополнительная форма представления – скелет, который обычно подчеркивает метрические и топологические свойства формы. Скелет является равноудалённой от границ тонкой версией фигуры. Иначе можно воспользоваться определением, основанным на понятии максимального пустого круга [13].

Пустой круг — это окружность, находящаяся в фигуре целиком включая внутренние точки. Пустой круг, который не содержится в других пустых кругах, является максимальным. Множество центров всех максимальных кругов фигуры называется скелетом. Для компактного расположения вершин(окружностей) навигационного графа как раз и нужны такие окружности как можно большего радиуса. Рассмотрим построение скелета на основе дистанционной карты и на основе диаграммы Вороного.

На дискретной плоскости нахождение точек скелета может быть выполнена на основе дистанционной карты. Дистанционная карта – это карта, в которой каждой точки дискретной фигуры ставит в соответствие расстояние до ближайшей границы. Для определения точек скелета используется некоторое локальное правило, если значение расстояния в окрестности

точки по любой из диагоналей, вертикали или горизонтали больше в данной точке, то она является точкой скелета. Такое правило может привести к разрывам и пропуску точек в скелете, поэтому необходимо включать дополнительную постобработку, которая заключается в склеивании отдельный частей скелета. Данный алгоритм относится к дискретным алгоритмам построения скелета.

Дадим описание диаграмме Вороного для того, чтобы было понятней построение скелета на ее основе. Диаграмма Вороного представляет собой разбиение плоскости на области, близкие к каждому из заданного набора объектов. В простейшем случае эти объекты представляют собой конечное число точек на плоскости (называемых начальными точками, узлами или сайтами) [14].

Пусть P – заданное на плоскости конечное множество точек, называемые сайтами. Для каждого сайта $p \in P$ существует соответствующая область, называемая ячейкой Вороного, состоящая из всех точек плоскости, расположенных ближе к этому сайту, чем к любому другому.

Рассмотрим множество из двух сайтов $P = \{p, q\}$, соединим эти точки и проведем срединный перпендикуляр полученного отрезка $[p, q]$, который называется бисектором. Получили две полуплоскости H_{pq} и H_{qp}, разделенных бисектором сайтов p и q, эти полуплоскости являются ячейками диаграммы Вороного (рисунок 1).

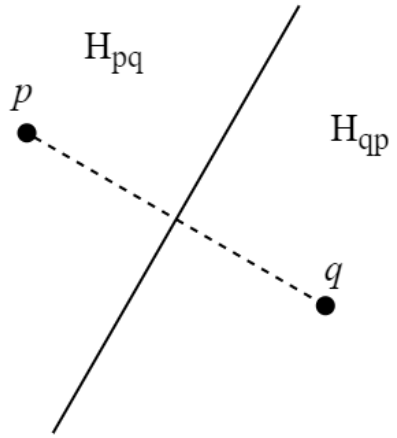

Рисунок 1 – Деление на полуплоскости

Если взять большее множество сайтов $P = \{p, q_1, \dots, q_n\}$, то ячейкой Вороного V_p для сайта p будет являться пересечением всех полуплоскостей H_{pq_i} (рисунок 2).

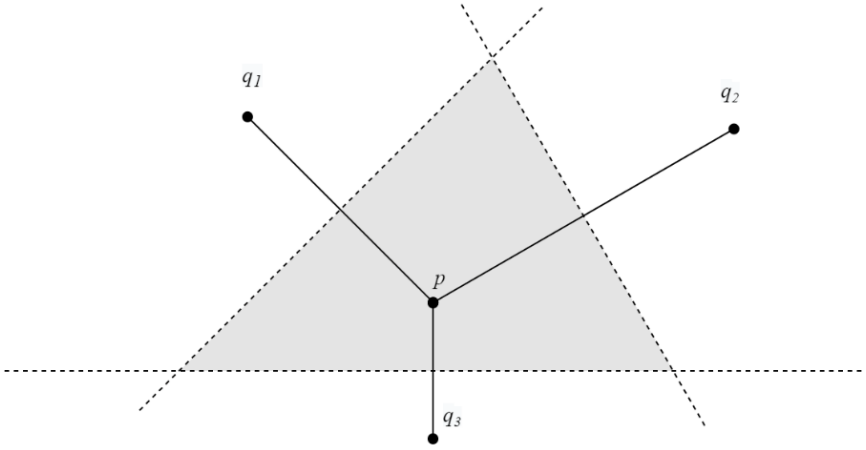

Рисунок 2 – Ячейка Вороного

Совокупность ячеек всех сайтов образуют диаграмму Вороного (рисунок 3). Алгоритм для нахождения каждой ячейки Вороного таким образом может быть реализован с вычислительной сложностью $O(n^4)$[13].

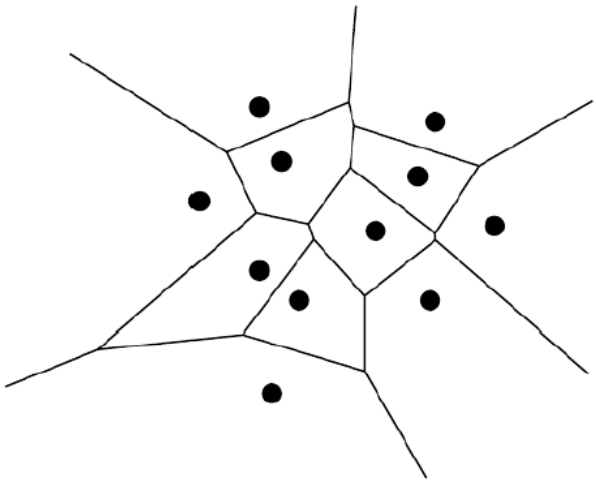

Рисунок 3 – Диаграмма Вороного

Если в качестве сайтов взять точки границы препятствий, то ребра диаграммы Вороного можно использовать в качестве элементов скелета двухмерной карты, т.к. данные ребра равноудалены от блажащих сайтов, но вместе с нужными ребрами диаграмма будет содержать еще «шумовые» ребра, которые не подходят для составления скелета (рисунок 4,а). Поэтому необходимо включить процесс исключения лишних ребер(«стрижка») диаграммы Вороного (рисунок 4,б).

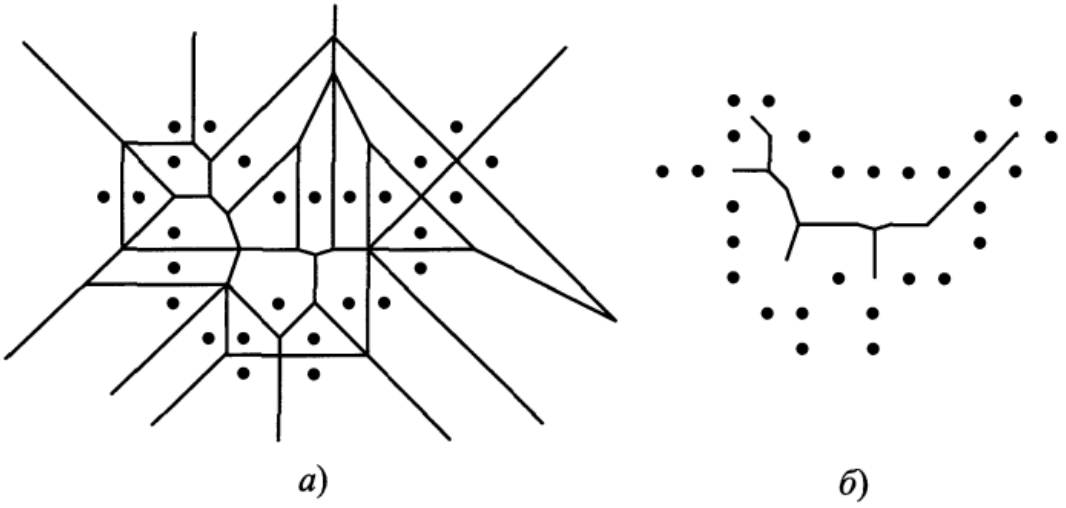

Рисунок 4 – Диаграмма Вороного граничных точек и соответствующий ей скелет без лишних ребер

Полученный после «стрижки» [13,15,16] скелет является непрерывным скелетом, в отличии от скелета, полученного с помощью, например, дистанционной карты.

Алгоритм Форчуна

Построение диаграммы Вороного множества точек на плоскости является сложным и трудоемким процессом. Для построение данной диаграммы можно использовать алгоритм Форчуна, являющимся одним из быстрых алгоритмов, он генерирует диаграмму Вороного из множества точек на плоскости за время $O(n * \log(n))$ [17]. Этот алгоритм основан на заметающей прямой, которая двигается слева направо. Заметающая прямая предоставляет собой вспомогательный объект, являющийся вертикальной прямой линией.

Основным недостатком является то, что вычисления выполняются с плавающей запятой вместо целых чисел, что открывает путь к ошибкам точности. Однако алгоритмы, использующие только целые числа, имеют риск переполнения целых чисел даже при достаточно малых ограничениях.

Напомним, что множество точек, равноудаленных от точки и прямой(директрисы), задается параболой. Теперь, если мы построим параболу для каждого сайта, где сайт является фокусом, а заметающая прямая является директрисой, все, что находится справа от всех парабол, будет неопределенной областью, а слева определённой. Граница областей определенности и неопределенности называется береговой линией, состоящей из параболических дуг. Каждая парабола будут меняться в зависимости от положения заметающей прямой относительно сайта — чем дальше она уходит от сайта вправо, тем больше расширяется парабола, однако в самом начале она вообще является лучом.

Двигая заметающую прямую слева направо, сохраняя структуру береговой линии, могут происходить следующие два типа событий, которые изменяют структуру береговой линии: событие точки и событие вершины.

В событии точки заметающая прямая обнаруживает сайт и вставляет ее параболическую дугу в береговую линию.

В событии вершины параболическая дуга исчезает с береговой линии в точке, соответствующей вершине диаграммы Вороного, просто удаляется дуга, длина которой уменьшается до нуля. Каждый раз, когда изменяется береговая линия, обновляются соответствующие точки для события вершины.

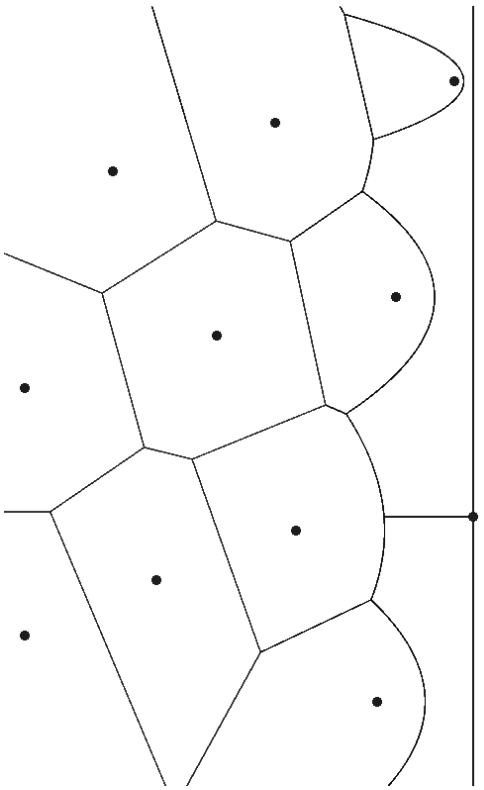

Рисунок 5 – Алгоритм Форчуна

Алгоритм Форчуна является не единственным алгоритмом построения с такой асимптотикой, но он достаточно понятен и не очень сложен в реализации.

Если препятствия на карте представлены многоугольниками, то можно воспользоваться алгоритмом «сегментных границ». Т.е. представить границу многоугольника в виде множества элементов – ее вершин и сторон. Стороной является отрезок границы с инцидентными ему вершинами. Сайтами в данном алгоритме являются стороны(сегменты) и вершины многоугольников. Алгоритм сегментный границ описан в [18].

Построение навигационного графа

Вершина графа создается из точки скелета, используя ее в качестве центра и соответствующий ей зазор(расстояние до ближайшей границы препятствия) в качестве радиуса. Точка скелета с максимальным зазором выбирается для создания вершины графа, и все точки скелета, покрытые этой вершиной, исключаются из выбора. Затем процесс повторяется до тех пор, пока не останется больше точек скелета для выбора. После нахождения всех вершин соединяем ребрами те из них, которые пересекаются. Пример навигационного графа изображен на рисунке 6.

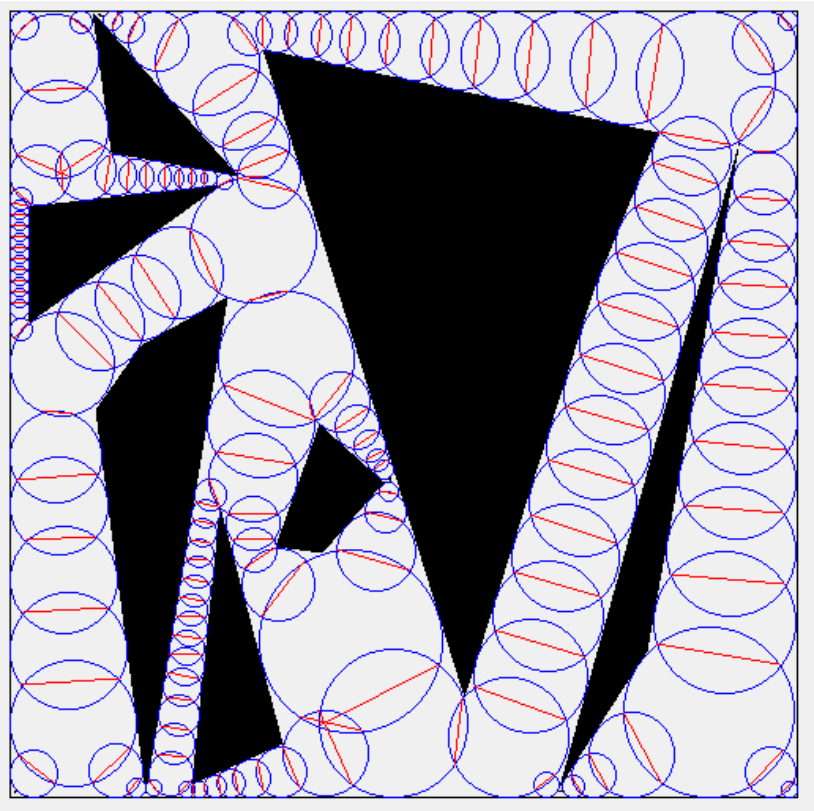

Рисунок 6 – Навигационный граф

Взвешивание графа

Для алгоритмов поиска на графах необходимо взвесить граф. Вес вершины – радиус. Вес ребра графа выбран векторным (две компоненты):

1) расстояние между центрами соседних вершин (оценка времени прохождения между вершинами);

2) ширина коридора (расчет может быть произведен по-разному, например, минимум из диаметров соседних вершин или же ширина области пересечения окружностей).

Исходя из заданной ширины перемещаемого объекта и весов ребер навигационного графа происходит исключение ребер, по которым объект не может перемещаться, и применяется один из алгоритмов поиска кратчайшего пути на графе.

Поиск на графе

Маршрут на навигационном графе может быть найден с помощью любого алгоритма поиска на графе, учитывающего веса ребер. Ниже рассмотрены примеры подходящих алгоритмов поиска.

Алгоритм Дейкстры решает задачу нахождения кратчайшего пути от исходной вершины на графе (источника) до пункта назначения [19]. Оказывается, что ко всем точкам графа можно найти кратчайшие пути от заданного источника за одно и то же время, поэтому эту проблему

иногда называют проблемой кратчайших путей с одним источником. Алгоритм работает только для графов без рёбер отрицательного веса.

Алгоритм Дейкстры включает следующие шаги:

1) Устанавливаются расстояние между всеми вершинами равное бесконечности, за исключением начальной вершины, в которой устанавливается расстояние равное 0.

2) Помечаются все вершины, включая начальную, в качестве не посещаемого узла.

3) Выбор из не посещённых вершин с наименьшим текущим весом (меткой вершины) в качестве текущей вершины «C».

4) Назовем «соседями» вершины, в которые ведут ребра из текущей вершины «C». Для каждого соседа «N» текущей вершины «C», кроме отмеченных как посещённые, добавляем метку равную сумме значения метки в текущей вершине «C» и веса ребра «C»-«N», если она меньше текущего значения в вершине «N».

5) Отмечаем текущий узел «C» как посещенный.

6) Необходимо повторить описанный выше шаги с шага 3 до тех пор, пока не будет посещен пункт назначения.

Алгоритм завершается после достижения всех вершин графа. В результате все посещенные вершины будут иметь метку, значение которой равно кратчайшему расстоянию от вершины источника до данный вершины. Могут быть ситуации, когда в графе есть вершины, до которых нельзя добраться, тогда алгоритм можно завершить досрочно.

Алгоритм A * похож на алгоритм Дейкстры, и единственное отличие состоит в том, что A * пытается найти «лучший» путь, используя эвристическую функцию «расстояние + стоимость» (обозначим как f(x)), которая отдает приоритет узлам, которые должны быть «лучше» других, в то время как алгоритм Дейкстры просто исследует все возможные пути. Функция f(x) — сумма двух других: функции стоимости достижения рассматриваемой вершины (x) из вершины источника (обозначим как g(x)), и функции эвристической оценки расстояния от рассматриваемой вершины к конечной (обозначим как h(x)) [20].

Обозначим веса вершин:

- G - стоимость передвижения из начальной вершины к данной вершине, следуя найденному пути к этой клетке;
- F – стоимость равная сумму G и функции h(x) для данной вершины.

Шаги алгоритма A*:

1) Формирование «открытого» списка (в начале работы алгоритма содержит вершину источник), в котором находятся вершины, которые необходимо просмотреть, и «закрытого» списка, который содержит просмотренный вершины.

2) Выбрать в качестве текущей вершины «C» вершину с наименьшим значением F и поместить в закрытый список.

3) Нахождение всех соседних вершин «N», который не входят в закрытый список.

4) Расчет для каждого соседа «N» временного G_TEMP равное сумме веса G текущей вершины «C» и веса ребра «C»-«N». Если «N» находится в открытом списке и его текущее G больше временного G_TEMP, или «N» не находится в открытом списке, то соответственно обновляем в отрытом списке или добавляем в отрытый список вершину «N», где G = G_TEMP и рассчитываем F, еще указываем для «N» вершину, из которой пришли(«C»).

5) Необходимо повторить описанный выше шаги с шага 2 до тех пор, пока не будет посещен пункт назначения.

Заключение

В статье были рассмотрены пути поиска маршрута площадного объекта с использованием навигационного графа. Рассмотрен процесс построения навигационного графа и выделены основные его этапы. Описаны вспомогательные методы ускорения построения навигационного графа (построение скелета и КД-дерева), а также приведены примеры подходящих алгоритмов поиска кратчайшего пути для данного графа.

Процесс построения навигационного графа, основанный на клеточной декомпозиции, является неточным, т.к. навигационный граф не захватывает все свободное от препятствий пространство, а вычисление графа использует дискретное представление окружающей среды. Поэтому для захвата узких проходов может потребоваться высокоточная сетка, что приводит к длительным расчетам.

В дальнейшем предстоит определить наиболее рациональный способ построения скелета и выбор вершин навигационного графа, как с точки зрения минимизации вычислительных затрат, так и с точки зрения максимального покрытия свободного пространства. Предстоит выбрать наиболее рациональный способ усечения навигационного графа в соответствии с габаритами объекта для дальнейшего поиска маршрута алгоритмами Дейкстры или A*.

Список литературы

1. ГОСТ 28441—99. Картография. Термины и определения / Внесён Госстандартом России. — Взамен ГОСТ 28441-90. — М.: ИПК Издательство стандартов, 2000. — С. 2. — 15 с. — (ГОСТ).
2. Цветков В. Цифровые карты и цифровые модели //Электронная версия размещается на сайте www. rae. ru. – 2000. – С. 348.
3. Абламейко С. В., Крючков А. Н. Информационные технологии создания и обновления цифровых и электронных карт местности //Информатика. – 2019. – №. 2 (02). – С. 86-93.
4. ГИС-ПАНОРАМА – Граф дорог [Электронный ресурс] URL: https://help.gisserver.ru/ru/panagro/index.html?roadgraph.html (дата обращения 10.01.22)
5. Судакова О. В. ИНТЕЛЛЕКТУАЛЬНАЯ ИНТЕРАКТИВНАЯ СИСТЕМА ПЛАНИРОВАНИЯ МАРШРУТОВ СЛУЖБ ДОСТАВКИ, ОСНОВАННАЯ НА ЗНАНИЯХ ПОЛЬЗОВАТЕЛЕЙ. РАЗРАБОТКА ГРАФА ДОРОГ //ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ В УПРАВЛЕНИИ, АВТОМАТИЗАЦИИ И МЕХАТРОНИКЕ. – 2021. – С. 259-264.
6. Mangalam K. et al. From goals, waypoints & paths to long term human trajectory forecasting //Proceedings of the IEEE/CVF International Conference on Computer Vision. – 2021. – С. 15233-15242.
7. Toma A. I. et al. Waypoint Planning Networks //arXiv preprint arXiv:2105.00312. – 2021.
8. Pettré J., Grillon H., Thalmann D. Crowds of moving objects: Navigation planning and simulation //ACM SIGGRAPH 2008 classes. – 2008. – С. 1-7.
9. Pettré J. et al. Real-time navigating crowds: scalable simulation and rendering //Computer Animation and Virtual Worlds. – 2006. – Т. 17. – №. 3-4. – С. 445-455.
10. Анатомия KD-Деревьев [Электронный ресурс] URL: https://habr.com/ru/post/312882/ (дата обращения 5.01.22)
11. K-d-дерево [Электронный ресурс] URL: https://ru.wikipedia.org/wiki/K-d-дерево (дата обращения 10.01.22)

12. Choi B. et al. Parallel SAH kD tree construction //High performance graphics. – 2010. – С. 77-86.
13. Местецкий Л. Непрерывная морфология бинарных изображений. Фигуры, скелеты, циркуляры. – Litres, 2018.
14. Erwig M. The graph Voronoi diagram with applications //Networks: An International Journal. – 2000. – Т. 36. – №. 3. – С. 156-163.
15. Ogniewicz R. L., Kübler O. Hierarchic voronoi skeletons //Pattern recognition. – 1995. – Т. 28. – №. 3. – С. 343-359.
16. Мерцалов А. Д. Сравнительный анализ алгоритмов построения скелетов бинарных изображений.
17. Fortune’s algorithm and implementation [Электронный ресурс] URL: http://blog.ivank.net/fortunes-algorithm-and-implementation.html (дата обращения 12.01.22)
18. Fortune S. A sweepline algorithm for Voronoi diagrams //Algorithmica. – 1987. – Т. 2. – №. 1. – С. 153-174.
19. Rachmawati D., Gustin L. Analysis of Dijkstra’s Algorithm and A* Algorithm in Shortest Path Problem //Journal of Physics: Conference Series. – IOP Publishing, 2020. – Т. 1566. – №. 1. – С. 012061.
20. A* Search Algorithm [Электронный ресурс] URL: https://www.geeksforgeeks.org/a-search-algorithm/ (дата обращения 12.01.22)

References

1. GOST 28441-99. Cartography. Terms and definitions / Introduced by the State Standard of Russia. - Instead of GOST 28441-90. - Moscow: IPK Publishing House of Standards, 2000. - p. 2. - 15 p. - (GOST).
2. Tsvetkov V. Digital maps and digital models //The electronic version is posted on the website www. rae. ru. - 2000. - p. 348.
3. Ablameyko S. V., Kryuchkov A. N. Information technologies for creating and updating digital and electronic maps of the area //Computer science. – 2019. – №. 2 (02). – pp. 86-93.
4. GIS-PANORAMA - Graph of roads [E-resource] URL: https://help .gisserver.ru/ru/panagro/index.html?roadgraph.html (Address date 10.01.22)
5. Sudakova O. V. INTELLIGENT INTERACTIVE ROUTE PLANNING SYSTEM FOR DELIVERY SERVICES BASED ON USER KNOWLEDGE. ROAD GRAPH DEVELOPMENT //INFORMATION TECHNOLOGIES IN MANAGEMENT, AUTOMATION AND MECHATRONICS. - 2021. - pp. 259-264.
6. Mangalam K. et al. From goals, waypoints & paths to long term human trajectory forecasting //Proceedings of the IEEE/CVF International Conference on Computer Vision. – 2021. – pp. 15233-15242.
7. Toma A. I. et al. Waypoint Planning Networks //arXiv preprint arXiv:2105.00312. – 2021.
8. Pettré J., Grillon H., Thalmann D. Crowds of moving objects: Navigation planning and simulation //ACM SIGGRAPH 2008 classes. – 2008. – pp. 1-7.
9. Pettré J. et al. Real-time navigating crowds: scalable simulation and rendering //Computer Animation and Virtual Worlds. – 2006. – Т. 17. – №. 3-4. – pp. 445-455.
10. Anatomy of KD-Trees [E-resource] URL: https://habr.com/ru/post/312882 / (Address date 5.01.22)
11. K-d-tree [E-resource] URL: https://ru.wikipedia.org/wiki/K-d-дерево (Address date 10.01.22)
12. Choi B. et al. Parallel SAH kD tree construction //High performance graphics. – 2010. – pp. 77-86.
13. Mestetsky L. Continuous morphology of binary images. Figures, skeletons, circulars. – Litres, 2018.

14. Erwig M. The graph Voronoi diagram with applications //Networks: An International Journal. – 2000. – Т. 36. – №. 3. – pp. 156-163.
15. Ogniewicz R. L., Kübler O. Hierarchic voronoi skeletons //Pattern recognition. – 1995. – Т. 28. – №. 3. – pp. 343-359.
16. Mertsalov A.D. Comparative analysis of algorithms for constructing skeletons of binary images.
17. Fortune's algorithm and implementation [E-resource] URL: http://blog.ivank.net/fortunes-algorithm-and-implementation.html (accessed 12.01.22)
18. Fortune S. A sweepline algorithm for Voronoi diagrams //Algorithmica. – 1987. – Т. 2. – №. 1. – pp. 153-174.
19. Rachmawati D., Gustin L. Analysis of Dijkstra's Algorithm and A* Algorithm in Shortest Path Problem //Journal of Physics: Conference Series. – IOP Publishing, 2020. – Т. 1566. – №. 1. – P. 012061.
20. A* Search Algorithm [E-resource] URL: https://www.geeksforgeeks.org/a-search-algorithm / (Address date 12.01.22)

Магомедов О. Р., Чернышёв С. А. Анализ существующих инструментов для кроссплатформенной разработки приложений// Международный журнал информационных технологий и энергоэффективности. – 2022. –
Т. 7 № 1(23) с. 17–24

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.4'2

АНАЛИЗ СУЩЕСТВУЮЩИХ ИНСТРУМЕНТОВ ДЛЯ КРОССПЛАТФОРМЕННОЙ РАЗРАБОТКИ ПРИЛОЖЕНИЙ

[1]Магомедов О. Р., [2]Чернышёв С. А.

[1,2]Санкт-Петербургский государственный экономический университет, Россия (191023, г. Санкт-Петербург, ул. Садовая, 21), e-mail: [1]smart7even@yandex.ru, [2]chernyshev.s.a@bk.ru.
[2]Санкт-Петербургский государственный университет промышленных технологий и дизайна, Россия (191186, г. Санкт-Петербург, ул. Большая Морская, 18)

В статье исследована область электронной коммерции как наиболее популярной и большой сферы, для которой разрабатываются приложения. Обоснована актуальность использования кроссплатформенных технологий, рассмотрены решения, позволяющие разрабатывать кроссплатформенные приложения.

Ключевые слова: электронная коммерция, кроссплатформенные технологии, кроссплатформенные фреймворки, разработка приложений.

ANALYSIS OF EXISTING TOOLS FOR CROSS-PLATFORM APPLICATION DEVELOPMENT

[1]Magomedov O. R., [1,2]Chernyshev S.A.

[1]Saint Petersburg state university of economics, Russian Federation, (191023, Saint-Petersburg, Sadovaya str. 21), e-mail: [1]smart7even@yandex.ru, [2]chernyshev.s.a@bk.ru.
[2]Saint Petersburg State University of Industrial Technologies and Design, Russian Federation (191186, Saint-Petersburg, Bolshaya Morskaya str. 18), e-mail: chernyshev.s.a@bk.ru.

The article investigates the field of e-commerce as the most popular and large area for which applications are developed. The relevance of using cross-platform technologies is substantiated, and solutions for developing cross-platform applications are considered.

Keywords: e-commerce, cross-platform technologies, cross-platform frameworks, application development.

Введение

Почти любой бизнес в 2022 году стремится развиваться в ИТ: создавать свои онлайн-магазины, создавать ПО для обеспечения работы предприятия, налаживания более эффективной коммуникации между сотрудниками, вести аккаунты в социальных сетях и так далее. Сфера ИТ развивается благодаря бизнесу, а сами информационные технологии позволяют бизнесу получать множество новых покупателей и увеличивать прибыль. Большинство приложений разрабатывается для E-commerce и для обеспечения E-commerce процессов. Поэтому рассмотрим E-commerce и особенности разработки, ориентированной на

E-commerce, а затем перейдем к анализу кроссплатформенных фреймворков, которые набирают популярность в разработке программных продуктов.

Электронная коммерция

E-commerce (электронная коммерция) - это сфера экономики, которая включает в себя все финансовые и торговые транзакции, осуществляемые при помощи компьютерных сетей, и бизнес-процессы, связанные с проведением таких транзакций [1]. E-commerce занимает большую часть рынка – по информации РБК на электронную торговлю в 2021 году будет приходиться 18,1% от всех розничных продаж в мире, а объем продаж составит почти 5 трлн. долларов [2]. Самыми яркими примерами и лидерами сферы являются такие компании как Alibaba, Amazon, eBay и ряд других. На вышеперечисленных площадках продаются и покупаются физические товары, но также есть сервисы, которые продают цифровые товары (Spotify и Apple music предлагают подписку на музыку, Netflix предлагает подписку на сериалы и кино), и сервисы по предоставлению услуг (Например, Booking.com предоставляет услугу по бронированию мест в отелях). Стоит отметить, что пандемия COVID-19 поспособствовала росту сферы E-commerce. При этом замечается тенденция, что в странах, у которых и ранее были развиты интернет-продажи, рост оказался незначительным относительно роста стран, в которых E-commerce находится на стадии становления (особенно Индия и Россия) [3].

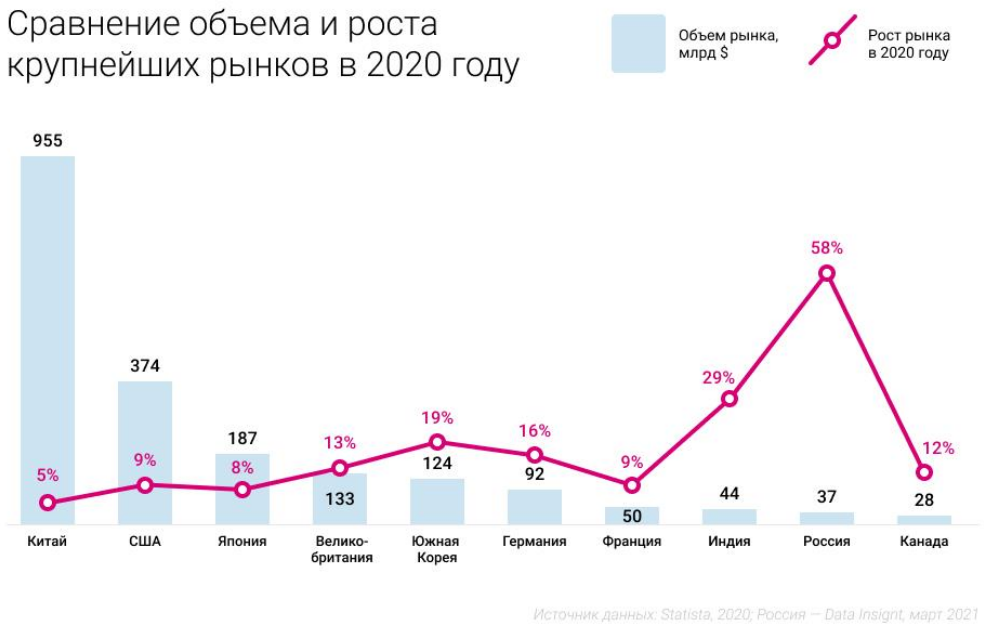

Рисунок 1 – Сравнение объема и роста крупнейших рынков E-commerce в 2020 году [3].

В соответствии с вышеперечисленным, E-commerce это обширная сфера, которая набирает обороты. Возможно, можно утверждать, что электронная коммерция создает новую культуру, когда в сознании людей формируется мысль о том, что купить товар или заказать

услугу можно, просто воспользовавшись устройством с выходом в Интернет, почти из любой точки мира.

Разработка E-commerce сервисов

Давайте рассмотрим вопрос с точки зрения разработки e-commerce сервисов, которых становится больше. Для того, чтобы создать успешную площадку, нужно разработать качественный и доступный программный продукт. Сервис должен быть доступен с любого устройства – телефона, планшета, ноутбука, компьютера, выдерживать большой поток пользователей, а также иметь красивый и удобный интерфейс. В основе e-commerce приложений в большинстве случаев используется клиент-серверная архитектура. В качестве клиентского приложения используются как web, так и мобильные приложения, а также приложения для персональных компьютеров. Большим плюсом веб приложения является то, что оно открывается с любого устройства – нужен лишь браузер. Но на практике у любого популярного e-commerce приложения есть хотя бы мобильное приложение, и в редких случаях приложение для персонального компьютера. Мобильное приложение как правило менее затратно по ресурсам, чем веб приложение, и дает больше возможностей для бизнеса и пользователя – позволяет отправлять push уведомления, хранить часть информации на телефоне, что позволяет пользоваться приложением даже оффлайн. Поэтому e-commerce приложение будет привлекательнее для пользователей и охватит большую аудиторию, если будет доступно на нескольких платформах (главные из которых Web, Android и iOS).

Кроссплатформенные решения

С точки зрения бизнеса поддерживать 3 различных платформы трудозатратно, потому что для каждой платформы нужна команда разработчиков, которая будет поддерживать проект. В настоящее время популярность набирают кроссплатформенные фреймворки, которые реализуют принцип «одна кодовая база – много платформ». Это означает, что написанное на таком фреймворке приложение компилируется в языки программирования, нативные для других платформ, и работает на различных платформах. Кроссплатформенные решения позволяют отказаться от необходимости писать отдельное приложение под каждую платформу. При описании кроссплатформенных фреймворков невозможно опустить тот факт, что они имеют некоторые ограничения. Каждая платформа имеет свои особенности взаимодействия с файловой системой, сетью, другими сервисами, и для обеспечения поддержки платформы все-таки нужны модули, написанные на нативных языках программирования. Однако разработчики фреймворков предоставляют возможность написания нативных модулей и предоставляют в своих решениях механизм взаимодействия кроссплатформенной кодовой базы с нативными модулями. Также стоит подчеркнуть, что благодаря развивающемуся сообществу разработчиков, использующих кроссплатформенные технологии, многие нативные модули уже написаны и лежат в открытом доступе. Таким образом, кроссплатформенные фреймворки, несмотря на свои ограничения, начинают занимать уверенное место среди технологий разработки клиентских приложений. Кроссплатформенных решений становится много и на этапе выбора технологий встает вопрос, какой кроссплатформенный фреймворк больше всего подходит для разрабатываемого

продукта. Для того, чтобы ответить на данный вопрос, проведем анализ и сравнение существующих фреймворков.

Анализ кроссплатформенных фреймворков

В настоящее время насчитывается более десятка кроссплатформенных фреймворков, но многие из них не пользуются популярностью среди разработчиков. Выделим наиболее используемые решения для анализа. В качестве критериев сравнения фреймворков выберем поддерживаемые платформы, удобство разработки, качество выходного продукта, развитость сообщества, а также сложность входа в разработку.

React Native – кроссплатформенный фреймворк, разработанный Facebook [5]. Фреймворк поддерживает все популярные платформы. Каждое React Native приложение работает в трех потоках. Основной поток обрабатывает отображение элементов пользовательского интерфейса и жесты пользователя, в то время как JavaScript поток выполняет JavaScript в отдельном движке, а за взаимодействие потоков отвечает отдельный модуль – bridge (мост). Третьим потоком является Shadow Thread, который отвечает за построение виртуального дерева пользовательского интерфейса для синхронизации состояния приложения с пользовательским интерфейсом. Во-первых, это позволяет использовать нативные для платформ элементы графического интерфейса, а для управления состояния приложением использовать возможности веб фреймворка React и JavaScript библиотек (менеджеров состояний, таких как Redux и Mobx, и других вспомогательных библиотек). Также React Native обладает технологией Hot reload, что значительно ускоряет разработку приложений. Hot reload («горячая замена») – обновление приложения без его полной перезагрузки. Когда разработчик добавляет элемент интерфейса или вносит другие изменения приложения, то происходит перестройка элементов пользовательского интерфейса, но приложение остается на той же странице и разработчик сразу видит изменения. Изучение фреймворка React Native будет несложным для фронтенд разработчиков, в особенности для тех, кто использовал в разработке web фреймворк React. Однако для перехода с нативной разработки будет необходимо освоить особенности языка JavaScript и концепции фреймворка. React Native применяется во многих приложениях, таких как Facebook, Instagram, Skype и Discord.

Xamarin является кроссплатформенным фреймворком Microsoft [6]. Приложение Xamarin строится на архитектуре MVVM (Model-View-ViewModel). View представляет из себя слой презентации, View-Model представляет из себя слой бизнес-логики, который осуществляет работу со слоем данных и обрабатывает пользовательские события, Model, слой данных, отвечает за получение данных из источников данных (файловой системы, Интернета, сервиса геолокации и других). Примечательно, что Xamarin предоставляет возможность сделать отдельно пользовательские интерфейсы под разные платформы с помощью библиотек Xamarin.Android и Xamarin.iOS, но также дает возможность декларативно описать один пользовательский интерфейс, который на этапе компиляции преобразуется в нативные элементы для каждой платформы. Для разработки на Xamarin необходимо владеть языком C#. На Xamarin пишутся приложения для различных сфер: доставки, банков, медицины, социальных сетей и других.

Kotlin Multiplatform Mobile (KMM) – это кроссплатформенный фреймворк, разработка компании JetBrains [7]. Фреймворк создан для мобильной разработки под iOS и Android. KMM устроен таким образом, что часть приложения, которая не зависит от платформы, пишется на Kotlin, а пользовательский интерфейс пишется на нативном языке платформы. К общему коду относится взаимодействие с сетью, сериализация и десериализация данных, бизнес-логика, модели данных, хранение данных, состояние приложения. За счет использования нативных элементов пользовательского интерфейса скорость работы приложений, написанных с использованием фреймворка, близка к скорости полностью нативных приложений. Механизм реализации платформозависимой логики устроен через интерфейсы. Объявляются функции, классы, интерфейсы, перечисления и свойства, которые должны быть реализованы для каждой платформы отдельно. Для написания программ на Kotlin Multiplatform Mobile необходимо знать язык Kotlin. Kotlin стал основным языком для разработки Android приложений, поэтому KMM будет несложен в освоении для Android разработчиков. Kotlin Multiplatform Mobile используется в достаточно широком списке компаний: Philips, Yandex, Netflix и других.

Flutter – кроссплатформенный фреймворк, разработанный Google для создания приложений под мобильные устройства, компьютеры и web [8]. Flutter позиционируется как фреймворк для быстрой разработки с выразительным и гибким интерфейсом и производительностью, не уступающей нативным приложениям. Flutter написан на языке Dart и приложения с использованием данного фреймворка тоже пишутся на Dart. Для отрисовки приложений на Flutter используется графический движок Skia, который предоставляет независимый от платформы интерфейс взаимодействия с ним. Flutter не использует нативные элементы графического интерфейса, но предлагает обширный список элементов (в Flutter они называются виджетами), которые максимально похожи на нативные элементы, хотя ими не являются. Flutter из коробки предоставляет разработчику 2 библиотеки виджетов – Material и Cupertino. Первая представляет собой набор элементов пользовательского интерфейса, реализующий систему дизайна Material, разработанную Google, вторая – виджеты, повторяющие дизайн элементов пользовательского интерфейса приложений Apple. Несмотря на использование для отрисовки пользовательского интерфейса отдельного движка, производительность приложений на Flutter близка к нативной. Разработка на Flutter ведется в режиме отладки, для замера производительности приложения и выявления проблемных мест используется режим Profile, а для запуска в эксплуатацию приложение компилируется под различные платформы и этот режим называется Release. Для отладки используются точки останова, которые можно поставить в любом месте исполняемого кода. Они позволяют просмотреть состояние программы и исполнить программу по шагам. Для просмотра границ виджетов и анимации в замедленном режиме, а также оценки производительности приложения используется инструменты разработки Dart. Они помогают понять, какая часть программы использует слишком много ресурсов, что может негативно сказываться на работе приложения. Запустившись в 2017 году, Flutter успел собрать вокруг себя быстро развивающееся сообщество разработчиков, которые помогают друг другу в освоении фреймворка, добавляют функциональность в фреймворк и исправляют ошибки и создают и поддерживают пользовательские библиотеки, в том числе библиотеки с виджетами, библиотеки, реализующие работу в фоновом режиме, работу с уведомлениями и другие. Для Dart сделана система управления библиотеками Pub. Разработчики библиотек могут

выкладывать свои библиотеки в систему, а пользователи могут использовать данные библиотеки. Для просмотра и поиска библиотек используется сайт pub.dev, на котором можно найти библиотеку и посмотреть ее документацию. У фреймворка также есть свой YouTube канал, на котором команда Flutter выпускает новости фреймворка, гайды, а также обзоры популярных библиотек и полезных виджетов. Некоторую сложность входа в разработку на Flutter может нести язык Dart. Этот язык программирования очень похож на другие строго-типизированные объектно-ориентированные языки программирования, но не похож на JavaScript, поэтому буден сложен в освоении для JavaScript разработчиков. С другой стороны Flutter очень похож на React Native по применяемым в фреймворке концепциям, основная из которых – дерево элементов пользовательского интерфейса. Стоит отметить, что в документации Flutter есть руководства для Android, iOS, React Native, web, Xamarin разработчиков для более плавного освоения фреймворка. Flutter для разработки своих приложений используют крупные компании: Google, eBay, Philips, BMW, Toyota и другие.

Таблица 1 - Сравнение кроссплатформенных фреймворков

	React Native	Xamarin	Kotlin Multiplatform Mobile	Flutter
Команда разработки	Facebook	Microsoft	JetBrains	Google
Удобство разработки	Hot reload, Dev tools, Режим отладки	Hot reload	Hot reload	Hot reload, Dev tools, Режим отладки
Сообщество	Большое сообщество	Большое сообщество	Развивающееся сообщество	Самое большое сообщество
Производительность	Высокая, близкая к нативной	Высокая, близкая к нативной	Высокая, близкая к нативной	Очень высокая
Отрисовка UI	Нативные элементы	Нативные элементы	Не занимается отрисовкой UI	Виджеты
Платформы	Android TV, iOS, macOS, tvOS, Web, Windows и UWP	Android, iOS, Windows, macOS	Android, iOS	Android, iOS, Linux, macOS, Web, Windows
Сложность входа	Легкая для JavaScript разработчиков, для остальных сложная	Легкая для Android разработчиков, для остальных средняя	Легкая для Android разработчиков, для остальных средняя	Легкая для React Native разработчиков, для остальных средняя

Заключение

В ходе рассмотрения кроссплатформенных фреймворков было выяснено, что самые популярные фреймворки обладают близкой друг к другу производительностью, хотя в разных задачах одни фреймворки несколько быстрее других [4], и инструментами разработки, и при выборе фреймворка необходимо прежде всего ориентироваться на то, какие языки и фреймворки знает команда разработки, насколько большое у него сообщество и насколько активно идет его поддержка. В статье были проанализированы самые удачные кроссплатформенные решения и каждое из них стоит того, чтобы к нему присмотреться, если перед вами стоит задача разработки приложений под несколько платформ.

Список литературы

1. А.В. Юрасов. Основы электронной коммерции: учеб. пособие – Горячая линия-Телеком, 2016. – 500 с.
2. Что такое e-commerce и как устроена онлайн-торговля [Электронный ресурс]. – Режим доступа: https://trends.rbc.ru/trends/industry/607fe4549a7947027eaffbe6
3. Как рынок eCommerce изменился за 2020 год, и какие настроения у потребителей в 2021 году? [Электронный ресурс]. – Режим доступа: https://www.metacommerce.ru/blog/ecommerce/#:~:text=7%20%D1%82%D1%80%D0%BB%D0%BD%20%D1%80%D1%83%D0%B1.-,%D0%B7%D0%B0%202020%20%D0%B3%D0%BE%D0%B4.,%D0%B3%D0%BE%D0%B4%D0%BE%D0%BC%2C%20%D1%80%D0%BE%D1%81%D1%82%20%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%B8%D0%BB%2078%25.
4. An empirical investigation of performance overhead in cross-platform mobile development frameworks [Электронный ресурс]. – Режим доступа: https://link.springer.com/article/10.1007/s10664-020-09827-6
5. React Native [Электронный ресурс]. – Режим доступа: https://reactnative.dev/
6. Xamarin [Электронный ресурс]. – Режим доступа: https://dotnet.microsoft.com/apps/xamarin
7. Kotlin Multiplatform [Электронный ресурс]. – Режим доступа: https://kotlinlang.org/docs/multiplatform.html
8. Flutter [Электронный ресурс]. – Режим доступа: https://flutter.dev/

References

1. A.V. Yurasov. Osnovy elektronnoj kommercii: ucheb. posobie – Goryachaya liniya-Telekom, 2016. – 500 p.
2. What is e-commerce and how Internet commerce works [Electronic resource]. - Available at: https://trends.rbc.ru/trends/industry/607fe4549a7947027eaffbe6.
3. How has the e-commerce market changed in 2020 and what is the consumer sentiment in 2021? [Electronic resource]. - Available at: https://www.metacommerce.ru/blog/ecommerce/#:~:text=7%20%D1%82%D1%80%D0%BB%D0%BD%20%D1%80%D1%83%D0%B1. -,%D0%B7%D0%B0%202020%20%D0%B3%D0%BE%D0%B4.,%D0%B3%D0%BE%D0

%B4%D0%BE%D0%BC%2C%20%D1%80%D0%BE%D1%81%D1%82%20%D1%81%D0%BE%D1%81%D1%82%D0%B0%D0%B2%D0%B8%D0%BB%2078%25.

4. An empirical study of performance outperformance in cross-platform mobile development frameworks [Electronic resource] - Available at: https://link.springer.com/article/10.1007/s10664-020-09827-6.
5. React Native [Electronic resource]. - Accessed at: https://reactnative.dev/.
6. Xamarin [Electronic resource]. - Available at: https://dotnet.microsoft.com/apps/xamarin
7. Kotlin Multiplatform [Electronic resource]. - Available at: https://kotlinlang.org/docs/multiplatform.html
8. Flutter [Electronic resource]. - Available at: https://flutter.dev/

Балашов О.В., Букачев Д.С. Искусственный интеллект в автоматизации процессов планирования и оперативного управления // Международный журнал информационных технологий и энергоэффективности. – 2022. –
Т. 7 № 1(23) с. 25–30

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 084.89

ИСКУССТВЕННЫЙ ИНТЕЛЛЕКТ В АВТОМАТИЗАЦИИ ПРОЦЕССОВ ПЛАНИРОВАНИЯ И ОПЕРАТИВНОГО УПРАВЛЕНИЯ

[1]Балашов О.В, [2]Букачев Д.С

[1]Смоленский филиал АО «Радиозавод», Россия, (214027, г. Смоленск, улица Котовского,2), e-mail: smradio@mail.ru
[2]ФГБОУ ВО Смоленский государственный университет, Смоленск, Россия (214000, г. Смоленск, ул. Пржевальского, 4), e-mail: dsbuka@yandex.ru

Предложена концепция интеллектуальной системы поддержки принятия решений, позволяющей решать задачи планирования предстоящих действий объектов организационно-технических систем, оценивания хода реализации ранее сформированного плана с возможностью его коррекции, а также интеграции рассматриваемой системы с внешними системами. Дальнейшее развитие предложенного подхода позволит разработать методологию построения интеллектуальных систем поддержки принятия решений, обеспечивающих автоматизацию процессов планирования и оперативного управления объектами организационно технических систем.

Ключевые слова: организационно-техническая система, искусственный интеллект, план предстоящих действий, управляющее решение, ситуация, ситуационный признак, действие, задача, автоматизированное планирование.

ARTIFICIAL INTELLIGENCE IN AUTOMATION OF PLANNING AND OPERATIONAL MANAGEMENT PROCESSES

[1]Balashov O.V., [2]Bukachev D.S.

[1]Smolensk branch of joint-stock company "Radio factory", Russia, (214027, Smolensk, street Kotovskogo, 2), e-mail: smradio@mail.ru
[2]Federal State Educational Institution of Higher Education Smolensk State University, Smolensk, Russia (214000, Smolensk, street Przewalski, 4), e-mail: dsbuka@yandex.ru

The concept of an intelligent decision support system is proposed, which allows solving the tasks of planning upcoming actions of objects of organizational and technical systems, evaluating the implementation of a previously formed plan with the possibility of its correction, as well as integrating the system under consideration with external systems. Further development of the proposed approach will allow us to develop a methodology for building intelligent decision support systems that automate the processes of planning and operational management of objects of organizational and technical systems.

Keywords: organizational and technical system, artificial intelligence, future action plan, control decision, situation, situational sign, action, task, automated planning.

Введение. К организационно-техническим системам (ОТС) относятся системы, на объектах которых производится принятие решений о выполнении мероприятий, согласованных с действиями других объектов (систем) или направленных на координацию

этих мероприятий. ОТС имеют иерархическую структуру. Функционирование ОТС (производственных, банковских, медицинских, строительных, военных систем и ряда других) происходит в условиях неопределенности информации, обрабатываемой при принятии решений. Неопределенность информации может быть вызвана ее неполнотой, избыточностью, недостоверностью, нечеткостью и неточностью [1, 3].

Условия функционирования ОТС, при которых происходит непрерывное изменение обстановки, приводящее к уникальности принимаемых решений, соответствуют условиям нестатистической неопределенности обрабатываемой информации. Для оценки решений принимаемых в этих условиях предлагается использовать методические средства теории возможностей [4, 5, 7].

Современные технологии автоматизированного планирования обладают рядом недостатков:

- отсутствие сформировавшегося подхода к полной автоматизации процессов принятия решений;
- отсутствие сформировавшегося подхода к автоматизированной разработке планов предстоящих действий, обеспечивающих функционирование организационно-технических систем (ОТС) и их интеграцию с другими ОТС;
- отсутствие сформировавшегося подхода к автоматизированной оценке хода реализации плана предстоящих действий и его коррекции.

Предлагаемая идея состоит в оснащении систем управления объектами ОТС интеллектуальными системами поддержки принятия решений (ИСППР). Ядро ИСППР должно включать в себя средства для выполнения следующих функций:

- представление цели функционирования объекта и ситуаций в формализованном виде;
- идентификация текущей ситуации функционирования объекта;
- ситуативный синтез моделей оценки принимаемых решений о выполнении задач;
- формирование рациональной стратегии управления объектом;
- разработка плана предстоящих действий;
- оценка хода реализации и автоматизированная коррекция плана предстоящих действий (при необходимости – перепланирование).

1. Постановка задачи.

Предлагаемая технология автоматизированного управления требует наличия на объектах различных ОТС интеллектуальных СППР (ИСППР) с типовым ядром, состоящим из средств вторичной и третичной обработки данных, которые работают по единым правилам. При этом допускается возможность реализации ИС на различных аппаратных платформах.

Целью разработки предлагаемой технологии автоматизированного управления является решение ряда системных задач, имеющих место при управлении ОТС. В качестве системных задач рассматриваются: разработка плана предстоящих действий ОТС; оценка хода реализации ранее сформированного плана и его коррекция; интеграция рассматриваемой системы с другими ОТС. Рассмотрим основное содержание этих задач, а также основные положения функционирования ядра ИСППР в процессе их решения.

Разработка планов предстоящих действий, оценка хода их реализации и коррекция производится по алгоритмам, единым для объектов различных ОТС. К данным алгоритмам относятся алгоритмы разработки плана предстоящих действий, оценки хода его реализации и коррекции, а также алгоритмы модификации предметной области. Последний алгоритм предназначен для непрерывного анализа имеющихся ресурсов системы и коррекции, в случае их изменения, перечней ситуационных признаков и управляющих решений, матриц соответствия и других данных образующих предметную область ИСППР.

Основные этапы процессов разработки плана предстоящих действий, оценки хода его реализации и коррекции следующие:

- идентификация текущей ситуации;
- формирование целевой ситуации;
- формирование стратегии управления;
- последовательная разработка виртуальных планов перехода системы из одной ситуации в другую;
- оценка возможностей по достижению целевой ситуации;
- выбор рационального варианта плана предстоящих действий;
- сравнение текущей ситуации с соответствующей ей ситуацией в ранее сформированном плане;
- оценка целесообразности сохранения целевой ситуации и/или стратегии управления;
- разработка новых планов перехода системы из одной ситуации в другую;
- формирование нового плана действий.

Интеграция с другими ОТС производится посредством согласования рассматриваемой системой своего плана предстоящих действий с аналогичными планами других систем. Отличительной чертой данного процесса от процесса разработки плана предстоящих действий является использование алгоритма модификации предметной области в процессе согласования планов предстоящих действий интегрирующихся ОТС. Модификация предметной области в данном случае состоит в увеличении количества и объемов баз данных, а также в расширении перечней ситуационных признаков и управляющих решений. Практическая реализация алгоритма модификации предметной области осуществляется логическими механизмами ядра ИСППР.

Под интеллектуальной системой объекта ОТС в рамках предлагаемой технологии понимается программная оболочка, обеспечивающая взаимодействие между ЭВМ и человеком, и предоставляющая ему возможности по управлению функциональной аппаратурой объекта при решении задач разработки плана предстоящих действий, оценки хода его реализации и коррекции. Основу этой оболочки образует типовое ядро ИСППР.

2. Предлагаемый подход.

Рассмотрим модульную структуру типового ядра ИССПР и опишем задачи, решаемые каждым модулем. Предлагаемая структура является развитием концепции структуры типового ядра ИССПР, изложенной в [2].

Основной задачей ***модуля идентификации текущей ситуации*** является управление модулями идентификации текущих значений ситуационных признаков (МИТЗСП), которые являются элементами прикладного программного обеспечения. Для работы модуля

предлагается усовершенствованный метод ситуационного управления. Проведенные авторами исследования позволили сделать вывод о возможности задания соответствия типа «значение ситуационного признака – множество управляющих решений». Правомерность данного подхода объясняется тем, что каждый ситуационный признак (СП) соответствует некоторой предметной области, в которой принимаются решения, зависящие от состояния объекта. Каждому СП соответствует конечное множество его возможных значений, а каждому значению – конечное множество решений о выполнении действий (задач).

Основной задачей ***модуля разработки плана предстоящих действий*** является разработка рационального плана предстоящих действий и его формализованное представление в памяти ЭВМ. Необходимым условием разработки плана является наличие текущей и целевой ситуаций. Идентификация текущей ситуации непрерывно производится соответствующим механизмом ядра ИСППР. Формирование целевой ситуации также производится соответствующим механизмом ядра ИСППР.

Задачей ***модуля формирования целевой ситуации*** является формализованное представление в памяти ЭВМ цели функционирования системы путем преобразования текста, речевого сообщения и/или мультимедийных данных в соответствующий их содержанию набор значений ситуационных признаков. В основе решения задачи лежит обработка ролевых ситуаций, состоящих из ролевых фреймов и фреймов понятий [6].

Модуль формирования стратегии управления предназначен для синтеза стратегии управления ОТС при ее переходе из текущей ситуации в целевую. В основе работы данного модуля лежит предположение о конечности множества ситуационных признаков, необходимых для формализованного представления состояния ОТС и условий её функционирования, а также значений этих признаков.

Модуль разработки плана перехода системы из одной ситуации в другую предназначен для разработки плана этого перехода и предоставляет ЛПР (человеку) возможности по решению следующих задач:

1) согласование управляющих решений, соответствующих текущим значениям различных ситуационных признаков;
2) ситуативный синтез виртуальных моделей реализации управляющих решений;
3) разработка и формализованное представление виртуального плана перехода системы из одной ситуации в другую;
4) оценка приоритетностей управляющих решений.

Модуль реализации плана предстоящих действий (оперативного управления) предназначен для решения следующих основных задач:

1) сравнение текущей ситуации с соответствующей ей ситуацией в ранее сформированном плане;
2) активация модели ранее сформированного плана перехода системы из одной ситуации в другую;
3) ситуативный синтез виртуальной модели плана перехода системы из текущей ситуации в требуемую;
4) оценка целесообразности сохранения стратегии управления и целевой ситуации;
5) интеграция системы с другими системами и ее самоорганизация в критических ситуациях.

Пользовательский интерфейс представляет собой программную оболочку, в состав которой входят:

- графический интерфейс, обеспечивающий активизацию модулей разработки и реализации плана предстоящих действий объекта;
- процедуры вывода экранных форм, соответствующих рассматриваемой ситуации;
- процедуры синтеза экранных форм;
- модули формирования, передачи и приема сообщений.

Система обработки логики управления предназначена для решения следующих задач:

- кодирование и декодирование логики обработки объектов данных;
- кодирование и декодирование логики плана предстоящих действий.

Управляющая система предназначена для управления работой ИСППР. Основными ее задачами являются управление процессами, оценка их приоритетности, контроль и управление вычислительными ресурсами. Управляющая система должна представлять собой совокупность логических модулей, обеспечивающих решение указанных задач. Алгоритмы работы управляющей системы должны быть адаптивными к технической оснащенности объекта (прежде всего конфигурацией и характеристиками вычислительной системы и системы передачи данных), а также к содержанию системного и прикладного программного обеспечения.

Заключение. Решение проблемы формализации процессов планирования и оперативного управления требует использования теоретических средств формализованного описания ОТС в текущих и прогнозируемых условиях обстановки. Одним из наиболее подходящих для этого средств является усовершенствованный метод ситуационного управления, который позволяет выполнить формализацию процессов планирования и оперативного управления.

Разработка теоретического подхода к автоматизации процессов планирования и оперативного управления, а также построение ядра ИСППР, позволяющих автоматизировать эти процессы, приводит к тесной интеграции технологий искусственного интеллекта и теории принятия решений. Дальнейшее развитие данного подхода позволит разработать методологию построения ИСППР, обеспечивающих автоматизацию процессов планирования и оперативного управления объектами ОТС.

Список литературы

1. Аверкин А.Н., Батыршин И.З., Блишун А.Ф. Нечеткие множества в моделях управления и искусственного интеллекта/ Под ред. Поспелова Д.А. - М .: Наука. Гл. ред. физ.-мат. лит., 1986. – 312 с.
2. Балашов О.В., Букачев Д.С. Подход к разработке технологии автоматизированного планирования и оперативного управления организационно-техническими системами // Международный журнал информационных технологий и энергоэффективности. – 2020. – Т. 5, № 4(18). – С. 21-32.
3. Борисов А.Н., Алексеев А.В., Меркурьева Г.В. и др. Обработка нечеткой информации в системах принятия решений. - М. : Радио и связь, 1989. – 304 с.

4. Борисов А.Н., Аппен Е.П. Оценка возможностных характеристик при анализе альтернатив. - В кн.: Методы принятия решений в условиях неопределенности. Рига: РПИ, 1980.
5. Дюбуа Д., Прад А. Теория возможностей//Приложения к представлению знаний в информатике. - М. :Радио и связь, 1990. – 287 с.
6. Минский М. Фреймы для представления знаний / Пер. с англ. О.Н. Гринбаума; Под ред. Ф.М. Кулакова. - Москва : Энергия, 1979. - 151 с. : черт. ; 20 см. - Библиогр.: с. 145-149
7. Ягер Р.Р. Нечёткие множества и теория возможностей. Последние достижения. Пер.с англ. М. : Радио и связь, 1986.

References

1. Averkin A.N., Batyrshin I.Z., Blishun A.F. Nechetkie mnozhestva v modelyah uprav-leniya i iskusstvennogo intellekta/ Pod red. Pospelova D.A. - M .: Nauka. Gl. red. fiz.-mat. lit., 1986. – 312 p.
2. Balashov O.V., Bukachev D.S. Podhod k razrabotke tekhnologii avtomatizirovannogo planirovaniya i operativnogo upravleniya organizacionno-tekhnicheskimi sistemami // Mezhdunarodnyj zhurnal informacionnyh tekhnologij i energoeffektivnosti. – 2020. – T. 5, № 4(18). – .PP 21-32.
3. Borisov A.N., Alekseev A.V., Merkur'eva G.V. i dr. Obrabotka nechetkoj informa-cii v sistemah prinyatiya reshenij. - M. : Radio i svyaz', 1989. – 304 p.
4. Borisov A.N., Appen E.P. Ocenka vozmozhnostnyh harakteristik pri analize al'-ternativ. - V kn.: Metody prinyatiya reshenij v usloviyah neopredelennosti. Riga: RPI, 1980.
5. Dyubua D., Prad A. Teoriya vozmozhnostej//Prilozheniya k predstavleniyu znanij v informatike. - M. :Radio i svyaz', 1990. – 287 p.
6. Minskij M. Frejmy dlya predstavleniya znanij / Per. s angl. O.N. Grinbauma; Pod red. F.M. Kulakova. - Moskva : Energiya, 1979. - 151 s. : chert. ; 20 sm. - Bibliogr.: PP. 145-149
7. YAger R.R. Nechyotkie mnozhestva i teoriya vozmozhnostej. Poslednie dostizheniya. Per.s angl. M. : Radio i svyaz', 1986.

Магомедов О. Р., Бодренков Г. А., Чернышёв С. А. Архитектура кроссплатформенных фреймворков // Международный журнал информационных технологий и энергоэффективности. – 2022. – Т. 7 № 1(23) с. 31–43

Международный журнал информационных технологий и энергоэффективности

Сайт журнала:
http://www.openaccessscience.ru/index.php/ijcse/

УДК 004.4'2

АРХИТЕКТУРА КРОССПЛАТФОРМЕННЫХ ФРЕЙМВОРКОВ

[1]Магомедов О. Р., [2] Бодренков Г. А., [3,4] Чернышёв С. А.
[1,2,3] Санкт-Петербургский государственный экономический университет, Россия, (191023, г. Санкт-Петербург, ул. Садовая 21), e-mail: [1],smart7even@yandex.ru, [2]glebbodrenkov@gmail.com, [3]chernyshev.s.a@bk.ru.
[4]Санкт-Петербургский государственный университет промышленных технологий и дизайна, Россия (191186, г. Санкт-Петербург, ул. Большая Морская, 18), e-mail: chernyshev.s.a@bk.ru

В статье исследована архитектура кроссплатформенных фреймворков. Рассмотрены концепции, возможности и ограничения архитектур наиболее популярных кроссплатформенных технологий, проведено сравнение архитектурных подходов рассмотренных решений, проанализированы текущие планы по развитию кроссплатформенных технологий.

Ключевые слова: кроссплатформенные технологии, кроссплатформенные фреймворки, архитектура.

ARCHITECTURE OF CROSS-PLATFORM FRAMEWORKS

[1]Magomedov O. R., [2] Bodrenkov G. A., [3,4] Chernyshev S. A.
[1,2,3] *Saint Petersburg state university of economics, Russian Federation, (191023, Saint-Petersburg, Sadovaya str. 21), e-mail: [1] smart7even@yandex.ru, [2] glebbodrenkov@gmail.com,[3] chernyshev.s.a@bk.ru.*
[4] *Saint Petersburg State University of Industrial Technologies and Design, Russian Federation (191186, Saint-Petersburg, Bolshaya Morskaya str. 18), e-mail: chernyshev.s.a@bk.ru.*

The article explores the architecture of cross-platform frameworks. The concepts, capabilities and limitations of the architectures of the most popular cross-platform technologies are considered, the architectural approaches of the considered solutions are compared, current plans for the development of cross-platform technologies are analyzed.

Keywords: cross-platform technologies, cross-platform frameworks, architecture.

Введение

Кроссплатформенные решения становятся все более популярными, так как позволяют разработать приложение, которое будет работать сразу на нескольких платформах. Это особенно важно для электронной коммерции, так как позволяет сократить сроки разработки и численность команды программистов и при этом сохранить поддержку всех широко используемых платформ. Кроссплатформенные фреймворки реализуют сложные архитектуры, и в них необходимо разбираться для того, чтобы понимать возможности и ограничения каждого решения. Многие концепции, используемые при разработке таких фреймворков, могут быть использованы и при разработке архитектур других программных продуктов. К самым популярных кроссплатформенным решениям относятся: Flutter, Xamarin, React Native и Kotlin Multiplatform.

Flutter

Первая версия Flutter SDK увидела свет в конце 2018 года, где Dart занял место основного языка программирования. Он был представлен как кроссплатформенный набор инструментов для разработки приложений с графическим пользовательским интерфейсом различной сложности под iOS и Android, позволяя напрямую взаимодействовать с базовыми сервисами целевых платформ. Постепенно список целевых платформ увеличивался и на данный момент времени разработка может вестись в рамках одной кодовой базы для: iOS, Android, macOS, Windows, Linux и Web. Конечно, это не значит, что код написанный под Android сразу будет работать в Web без доработок, но сама концепция и приложенные к её реализации усилия Google способствуют тому, чтобы пристально присмотреться к этому инструменту.

Архитектурные слои

Архитектура Flutter для всех платформ, за исключением Web, состоит из трех слоев (рисунок 1 [1]):

1. Flutter framework;
2. Flutter engine;
3. Embedder.

Каждый из представленных слоев имеет некоторое количество независимых библиотек, которые расположены на различных уровнях и спроектированы таким образом, что библиотека более верхнего уровня имеет зависимость только от библиотек или библиотеки следующего за ней нижнего уровня. Такой подход делает любую из библиотек заменяемой.

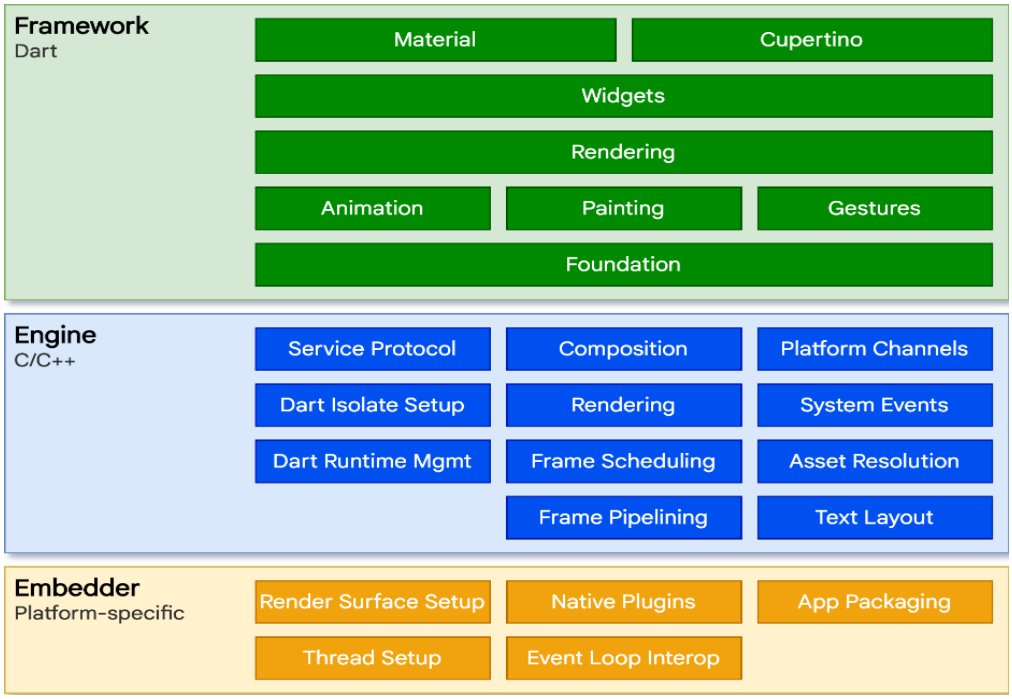

Рисунок 1 – Архитектура Flutter SDK

Слой *Embedder* позволяет упаковывать разрабатываемые приложения на Flutter под различные целевые платформы таким образом, как будто они были написаны на поддерживаемом ими языке программирования. Таким образом, слой *Embedder* обеспечивает для приложения «точку входа» и отвечает за взаимодействие с целевой операционной системой для доступа к её службам (визуализация, ввод/вывод и т.д.) и управляет циклом обработки сообщений. Для каждой из поддерживаемых платформ этот слой написан на подходящем к ней языке программирования:

- Java/Kotlin и C++ для Android;
- Objective-C и Swift для iOS и macOS;
- C++ для Windows и Linux.

Именно благодаря слою *Embedder* разрабатываемое на Flutter приложение может быть как самостоятельным, так и интегрироваться в качестве модуля в нативные приложения.

Слой *Flutter engine* в основном написан на C++ и обеспечивает низкоуровневую реализацию API Flutter: работа с графикой (графический движок Skia), операции ввода-вывода, межсетевое взаимодействие и т.д., а также отвечает за поддержку ряда специальных возможностей, среду выполнения и компиляции Dart. Именно благодаря собственному графическому движку Skia, который отвечает за отрисовку пользовательского интерфейса на любой из платформ, Flutter SDK не пошел по стопам React Native. То есть все виджеты отрисовываются средствами Flutter и нет никаких дополнительных прослоек, что сказывается на плавности их отрисовки, которое не уступает нативному.

Чаще всего разработчики и не подозревают о наличии рассмотренных ранее слоев и взаимодействуют только со слоем *Flutter framework*, написанным на Dart. Он включает в себя довольно богатый набор базовых библиотек, расположенных на различных уровнях текущего слоя. Давайте перечислим существующие уровни слоя *Flutter framework*, начиная с самого нижнего:

- Уровень *Foundational* предоставляет базовые классы и функции, которые используются для создания приложения, а также содержит API-интерфейсы для связи со слоем *Flutter engine*.
- Уровень *Rendering* обеспечивает абстракцию для работы с компоновкой виджетов, позволяет построить дерево визуализируемых объектов и динамически управлять ими, что автоматически отразится на структуре дерева.
- Уровень *Widgets* представляет собой композицию абстракций и модель реактивного программирования. Так, например, у каждого объекта на уровне рендеринга имеется соответствующий класс на уровне *Widgets*, что позволяет определять повторно используемые комбинации классов.
- Библиотеки *Material* и *Cupertino* предоставляют разработчику наборы элементов управления, которые используют примитивы композиции уровня *Widgets* для реализации виджетов в стиле Material или iOS.

Как в Dart существует концепция, что все является объектом, так и во Flutter имеется своя концепция: все является виджетом. То есть графический пользовательский интерфейс разрабатываемых приложений полностью состоит из виджетов и их различной компоновки. Каждому виджету соответствует свой элемент на уровне *Rendering*, в соответствии с чем на этапе сборки Flutter переводит виджеты, которые используются в коде, в соответствующее дерево элементов (рисунок 2 [1]):

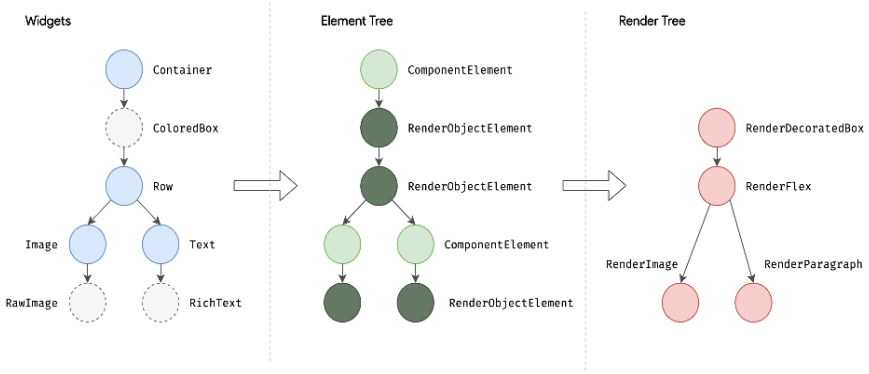

Рисунок 2 – Приведение дерева виджетов к дереву рендеринга

Дерево виджетов отвечает за конфигурирование, а именно: декларативное описание пользовательского интерфейса и хранение свойств виджетов. Дерево элементов отвечает за управление, то есть элементы управляют жизненным циклом виджетов, а также осуществляют их связывание в древовидную иерархию и с объектами рендеринга. Дерево рендеринга отвечает за отрисовку виджетов с учетом их положения и ограничений.

Архитектура Web

Еще с момента своего появления Dart компилировался в JavaScript из-за чего портирование Flutter для Web пошло по другому пути. Это связано с тем, что слой *Flutter engine* написан на C++ и ориентирован на взаимодействие с операционной системой, а не браузером (рисунок 3 [1]).

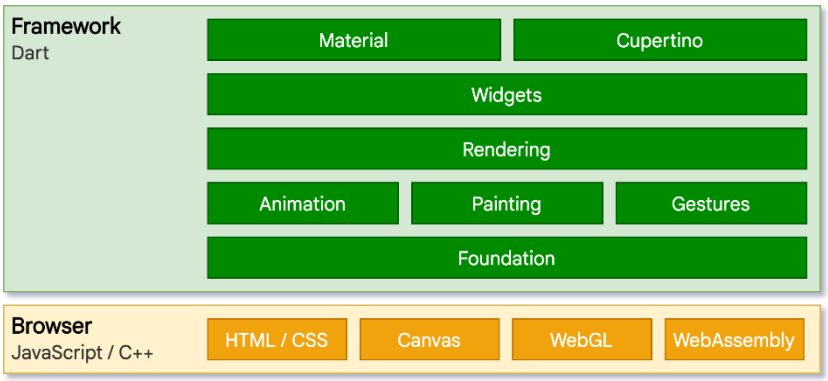

Рисунок 3 – Архитектура Flutter Web

Flutter Web обеспечивает повторную реализацию слоя *Flutter engine* поверх стандартных API-интерфейсов браузера. В связи с этим существует два способа рендеринга содержимого Flutter для web: HTML и WebGL. В первом случае Flutter использует HTML, CSS, Canvas и SVG. А для рендеринга в WebGL Flutter использует версию Skia (CanvasKit), скомпилированную в WebAssembly.

Flutter: Итог

Таким образом, можно утверждать, что архитектура Flutter хорошо спроектирована. Она позволяет улучшать фреймворк и добавлять поддержку любых платформ. Архитектура разделена на слои, что позволяет однозначно установить иерархию зависимостей и сделать компоненты фреймворка заменяемыми. К минусу можно отнести использование Skia, потому что пользовательский интерфейс, хоть и мимикрирует под интерфейс нативных платформ, но в некоторых деталях заметно отличается от него. Фреймворки, рассмотренные далее используют нативные элементы пользовательского интерфейса, но в связи с этим вынуждены реализовывать более сложную архитектуру для установления двусторонней связи между фреймворком и нативной платформой.

Xamarin

Xamarin — это платформа с открытым исходным кодом, предназначенная для построения современных производительных приложений для iOS, Android и Windows с .NET. [2] Платформа Xamarin представляет собой уровень абстракции, который обеспечивает взаимодействие между общим кодом и кодом нативной платформы.

Благодаря Xamarin в среднем 90 % кода приложения может использоваться без изменений на разных платформах. Разработчик может написать всю бизнес-логику на одном языке, а также получить близкую к нативной производительность приложения и графический пользовательский интерфейс, которые характерны для целевой платформы.

Платформа Xamarin ориентирована на разработчиков, перед которыми стоят следующие задачи:

- Совместное использование кода, тестов и бизнес-логики на различных платформах;
- Написание кроссплатформенных приложений на языке C#.

Архитектура

Структуру платформы Xamarin можно представить следующим образом (рисунок 4 [2])

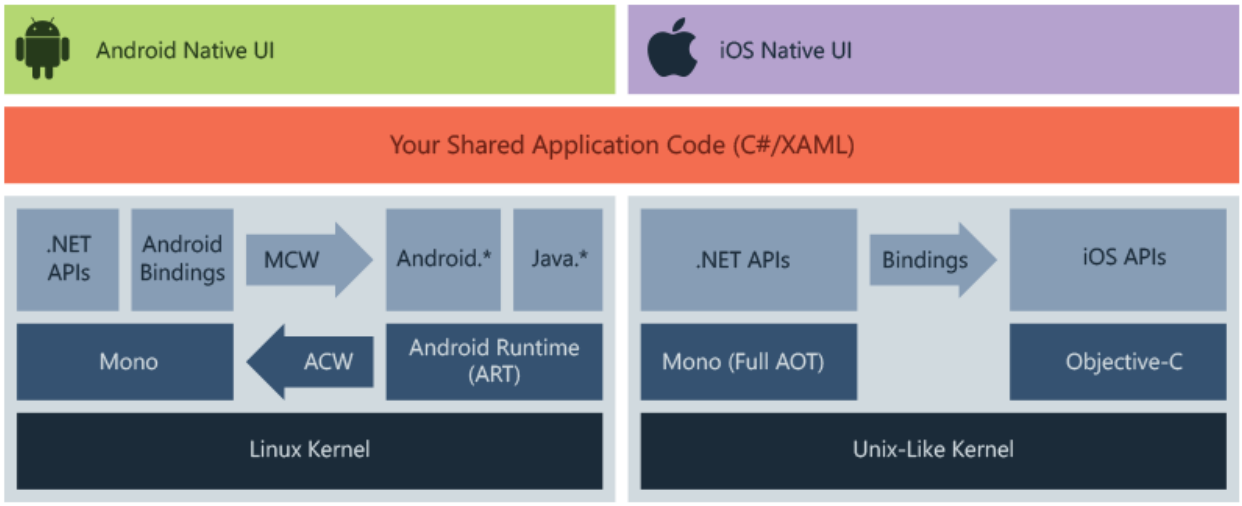

Рисунок 4 – Архитектура Xamarin

Xamarin работает поверх фреймворка Mono, который предоставляет opensource-реализацию .NET Framework [3] и может работать с разными платформами, такими как Linux, MacOS и т.д.

На уровне каждой отдельной платформы Xamarin полагается на ряд субплатформ. В частности:

- Xamarin.Android - библиотеки для создания приложений на ОС Android;
- Xamarin.iOS - библиотеки для создания приложений для iOS.

Через них приложения могут направлять запросы к прикладным интерфейсам на устройствах под управлением ОС Android или iOS.

Xamarin.Android

Приложения Xamarin.Android компилируются из языка C# в промежуточный язык, который при запуске приложения проходит Just-in-Time-компиляцию (рисунок 5 [2]). Приложения Xamarin.Android работают в среде выполнения Mono параллельно с виртуальной машиной среды выполнения Android (ART). Xamarin предоставляет привязки .NET к пространствам Android и Java, а также обращается к этим пространствам с использованием управляемых оболочек (ACW) и предоставляет ART их Android (ACW), благодаря чему обе среды могут вызывать код друг друга.

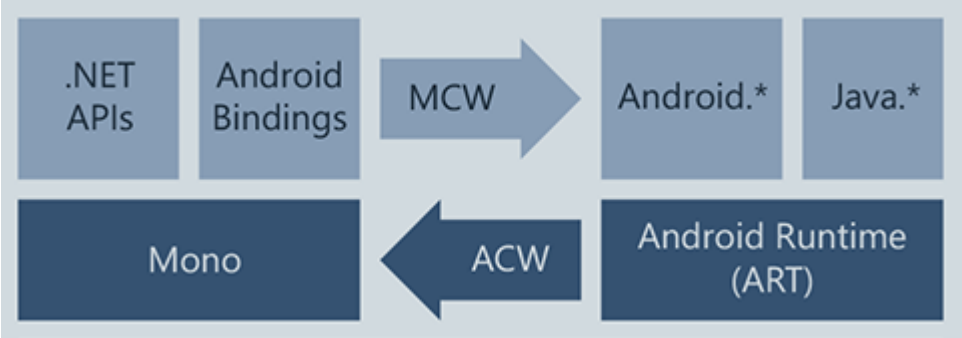

Рисунок 5 – Архитектура Xamarin.Android

Xamarin.iOS

Код приложений Xamarin.iOS собирается под целевую платформу посредством Ahead-of-Time-компиляции из языка C#. Xamarin использует селекторы, чтобы предоставить Objective-C управляемому коду C# и управляемый код C# для Objective-C. Селекторы и регисторы в совокупности называются "привязками" и обеспечивают взаимодействие между Objective-C и C# (рисунок 6 [2]).

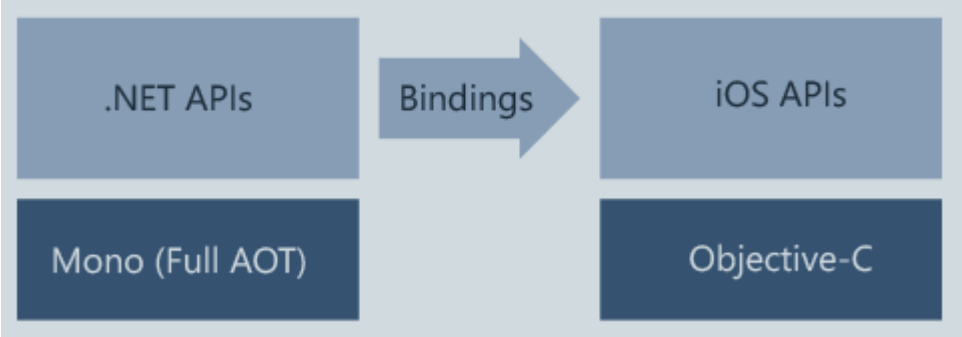

Рисунок 6 – Архитектура Xamarin.iOS

В итоге благодаря этим платформам можно отдельно создавать приложения для Android или iOS. Наиболее важной особенностью Xamarin является возможность создавать кроссплатформенные приложения - одна логика на все платформы. Для этого используется технология Xamarin.Forms, которая является слоем абстракцией над Xamarin.Android и Xamarin.iOS. С помощью Xamarin.Forms определяется визуальный интерфейс и выполняется привязка к нему бизнес-логики на C#, которая одинаково будет работать на Android, iOS и Windows.

.NET MAUI

.NET Multi-platform App UI (.NET Кроссплатформенное приложение пользовательского интерфейса) — это кроссплатформенный фреймворк, позволяющий создавать мобильные и компьютерные приложения при помощи C# и XAML [3]. В данный момент (февраль 2022 года) он находится в активной разработке и доступна версия Preview 12. Позиционируется как “эволюция Xamarin.Forms” и поддерживает миграцию приложения из этого фреймворка с небольшими изменениями в коде. Оба фреймворка похожи друг на друга, но есть одно главное отличие - в одном проекте .NET MAUI можно будет создавать кроссплатформенные приложения и добавлять специфический для платформы код непосредственно там, где это необходимо. Одна из основных целей этого фреймворка - реализовать как можно больше исходной логики приложения и макета пользовательского интерфейса в основном коде, еще больше, чем позволяет Xamarin.Forms.

Xamarin: Итог

Xamarin позволяет довольно просто создавать, поддерживать и обновлять кроссплатформенное программное обеспечение, при этом оставляя сравнимую с нативной производительность. Главными минусами Xamarin на данный момент являются:

- желательное наличие начальных знаний в языках нативных платформ для успешного использования возможностей каждой из них;
- чаще всего, приложения на Xamarin весят больше, чем приложения на нативных платформах. На рисунке 7 показано, сколько объема памяти занимает приложение, выводящее «Hello World»;
- Xamarin.Forms в ближайшее время будет заменено .NET MAUI.

Рисунок 7 – Распределение объема памяти приложения «Hello World»

React Native

React Native – кроссплатформенный фреймворк с открытым исходным кодом, разработанный Facebook. Фреймворк поддерживает все популярные платформы: Android, Android TV, iOS, macOS, tvOS, Web, Windows. Изучение фреймворка React Native будет несложным для фронтенд разработчиков, в особенности для тех, кто использовал в разработке web-фреймворк React.

Архитектура

Фундаментальным понятием в архитектуре React является Virtual DOM (виртуальное дерево элементов пользовательского интерфейса) - это абстракция, которая держит представление UI в памяти в виде дерева компонентов и синхронизирует его с пользовательским интерфейсом.

Каждое React Native приложение работает в трёх потоках [4]. Поток пользовательского интерфейса обрабатывает отображение элементов пользовательского интерфейса и жесты пользователя. JavaScript поток отвечает за управление состоянием приложения: получение, передачу и обработку данных - и построение Virtual DOM при изменении состояния приложения. Далее JavaScript поток передает его Shadow потоку, который работает в фоновом режиме и отвечает за вычисление изменений пользовательского интерфейса и уведомление потока пользовательского интерфейса о том, когда и где нужна перерисовка пользовательского интерфейса. Именно введение такой абстракции как Virtual DOM позволяет отслеживать те части пользовательского интерфейса, в которых нужны изменения, и производить только их перерисовку, не перерисовывая приложение целиком. React Native предоставляет набор абстрактных компонентов пользовательского интерфейса (View), которые преобразуются в нативные элементы каждой платформы при компиляции приложения. А также предоставляет универсальный API взаимодействия с такими внешними модулями как: сеть, файловая система и т.д.

Архитектура, представленная в React Native, имеет ряд преимуществ:

1. Она позволяет осуществить разделение процесса обновления пользовательского интерфейса на два этапа: reconciliation (согласование) и rendering (отрисовка). Такое разделение дает возможность хранить представление пользовательского интерфейса в памяти и при изменении состояния приложения на этапе reconciliation сравнивать предыдущее и текущее состояния и посылать модулю отрисовки команды на обновление только тех частей пользовательского интерфейса, для которых изменилось состояние. Такой механизм уже применялся в веб фреймворке React и был перенесен в React Native.
2. Она предусматривает использование нативных для платформ элементов графического интерфейса, но отделение этапов rendering и reconciliation позволяет также создать свой движок для управления отрисовкой пользовательского интерфейса, что

продемонстрировано в реализации библиотеки react-native-skia [5], которая позволяет использовать графический движок skia. Библиотека находится на стадии Proof of Concept (Доказательства концепции), но сам тот факт, что архитектура React Native настолько гибка, что позволяет подменять реализации модулей для отрисовки пользовательского интерфейса, неоспорим.

3. Использование JavaScript как языка программирования позволяет использовать API веб фреймворка React и JavaScript библиотек (менеджеров состояний, таких как Redux и Mobx, и других вспомогательных библиотек).

React Native: Итог

React Native стремительно развивается и постепенно решает архитектурные проблемы. Еще недавно к минусам архитектуры можно было отнести использование модуля bridge для взаимодействия потоков, что ограничивало быстродействие приложений, написанных на React Native. Сообщения для обмена информацией между потоками передавались через bridge в формате JSON. Это добавляло необходимость сериализовать и десериализовать данные, что замедляло приложение. Новый отрисовщик интерфейса улучшает обмен данными с помощью доступа к состоянию JavaScript потока напрямую используя JavaScript интерфейсы (JSI).

Тем не менее «узким местом» React Native до сих пор остается однопоточность JavaScript, что затрудняет реализацию сложных вычислений на устройстве клиента. Запуск сложного вычисления приведет к зависанию пользовательского интерфейса. Обход данного ограничения состоит в использовании процессов, но такое решение требует реализовывать межпроцессное взаимодействие, так как память процессов по умолчанию изолирована. Положительным моментом является то, что сообщество разрабатывает библиотеку для межпроцессного взаимодействия react-native-multithreading [6].

Kotlin Multiplatform

Kotlin Multiplatform (KM) [7] – это технология, позволяющая писать приложения под разные платформы с использованием Kotlin, разработанная компанией JetBrains. Kotlin Multiplatform включает в себя два фреймворка: Kotlin Multiplatform Mobile (KMM) [8] и Compose Multiplatform [9]. KMM создан для мобильной разработки под iOS и Android, а Compose Multiplatform включает поддержку Desktop платформ (Windows, macOS, Linux) и Web. Кроссплатформенная разработка с использованием Kotlin Multiplatform устроена таким образом, что часть приложения, которая не зависит от платформы, пишется на Kotlin, а пользовательский интерфейс пишется на нативном языке платформы с использованием фреймворков Compose Multiplatform и Kotlin Multiplatform Mobile (рисунок 8 [7]). К общему коду относится взаимодействие с сетью, сериализация и десериализация данных, бизнес-логика, модели данных, хранение данных, состояние приложения и т. д.

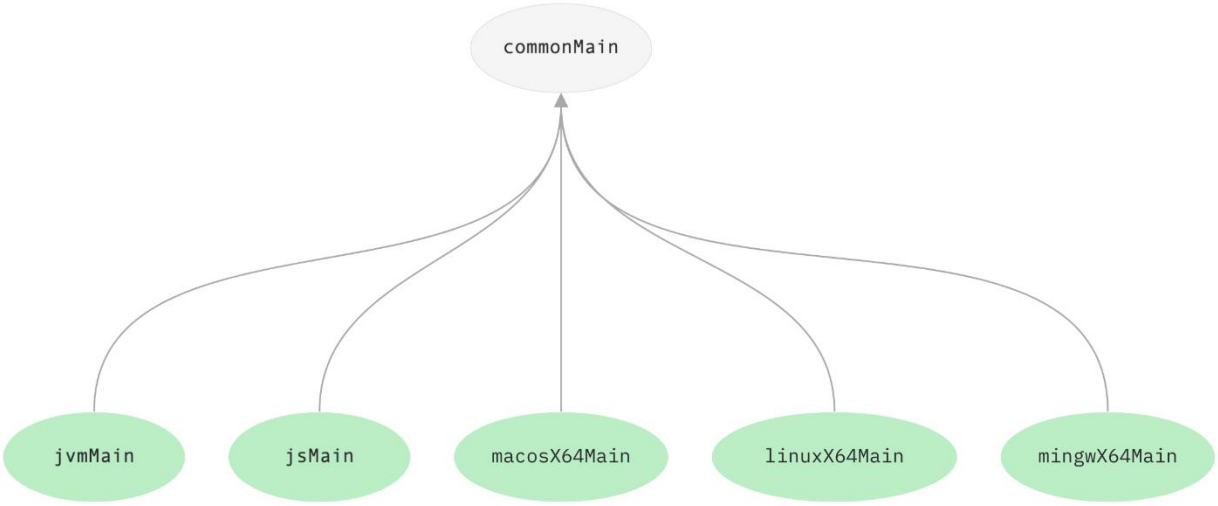

Рисунок 8 – Разделение кода приложения на общую и платформозависимую логику

За счет использования нативных элементов пользовательского интерфейса скорость работы приложений, написанных с использованием фреймворков, близка к скорости полностью нативных приложений. Механизм реализации платформозависимой логики устроен через интерфейсы. Объявляются функции, классы, интерфейсы, перечисления и свойства, которые должны быть реализованы для каждой платформы отдельно. Затем они реализуются для каждой платформы с использованием платформенных версий Kotlin, которые имеют доступ к нативному коду платформы, что позволяет использовать все нативные возможности (рисунок 9 [7]).

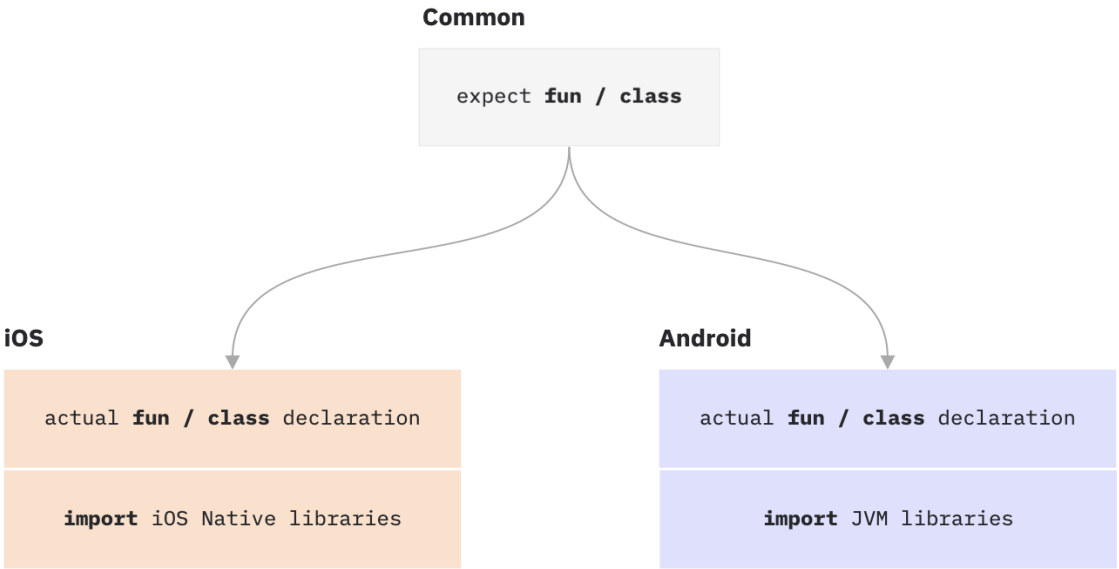

Рисунок 9 – Объявление и реализация платформозависимого интерфейса приложения

При запуске и сборке приложения с использованием Kotlin для Web, код на Kotlin транспилируется в JavaScript транспилятором Kotlin/JS. При использовании Kotlin для Android код компилируется в байт-код, исполняемый далее на Java Virtual Machine. При использовании Kotlin для других платформ (macOS, iOS, tvOS, watchOS, Linux, Windows) код приложений компилируется в нативные бинарные файлы, которые запускаются без виртуальной машины (рисунок 10 [7]).

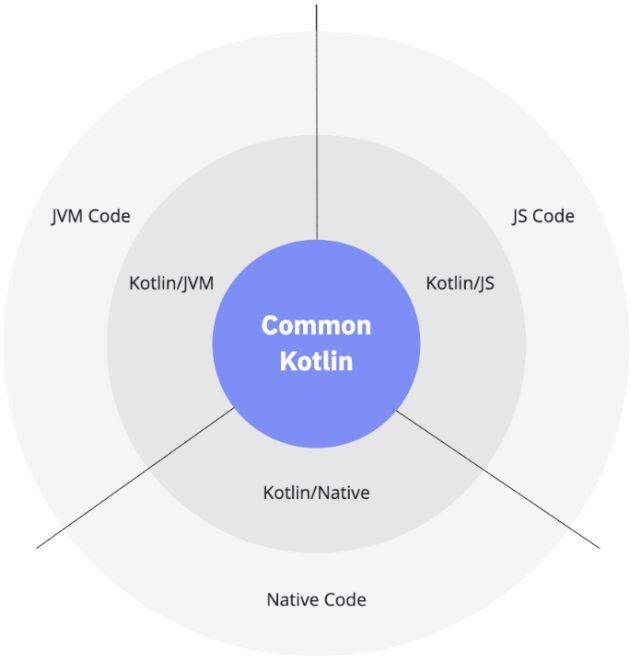

Рисунок 10 – Транспиляторы и компиляторы Kotlin

Kotlin Multiplatform: Итог

Являясь достаточно молодой технологией, Kotlin Multiplatform представил свое видение кроссплатформенности, проводя границу между пользовательским интерфейсом и бизнес-логикой приложения. Логика приложения выступает ядром, вокруг которого добавляется поддержка платформ с помощью реализации пользовательского интерфейса посредством двух технологий - Kotlin Multiplatform Mobile и Compose Multiplatform. Большим плюсом KM является его компилируемость в нативный код платформ, что дает возможность не включать виртуальную машину исполнения Kotlin в релизную сборку приложения.

Сравнение исследуемых фреймворков

У Представленных выше фреймворки есть как общие концепции, так и различия.

К общему подходу, использующемуся во всех представленных выше фреймворках, относится использование деревьев для построения пользовательского интерфейса. Фреймворки используют их для оптимизации рендеринга, что позволяет обновлять пользовательский интерфейс только тех компонентов, для которых изменилось состояние.

Необходимо выделить, что у фреймворков разный подход к пользовательскому интерфейсу. React Native, Xamarin и Kotlin Multiplatform используют нативные компоненты

для отрисовки пользовательского интерфейса, в то время как Flutter использует свой 2D движок Skia.

Также различны и способы взаимодействия с нативной платформой. Kotlin Multiplatform компилируется в нативный код, а Flutter, React Native и Xamarin запускаются в отдельном ядре и коммуницируют с ядром пользовательского интерфейса для предоставления ему состояния приложения и обработки событий пользовательского интерфейса. В связи с этим в сборку приложений данных фреймворков входят среды исполнения языков программирования, для Flutter это среда исполнения Dart, React Native - JavaScript, Xamarin - C#. Как правило среды исполнения занимают много места в памяти, поэтому сборка таких приложений становится больше.

Следует отметить, что первая стабильная версия Compose Multiplatform вышла только 12 декабря 2021 года, поэтому принимать решение об использовании данного фреймворка для больших проектов нужно с осторожностью, в то время как Xamarin, Flutter и React Native уже «прижившиеся» технологии, которые используются в продуктах ряда больших компаний.

Отметим развитие фреймворков. Flutter и React постепенно развиваются, выпуская новые версии, которые улучшают производительность, удобство разработки, устраняют ошибки. А Xamarin помимо этого эволюционирует в MAUI, что позволяет добавить поддержку десктопных операционных систем. Эта эволюция - хороший шаг в развитии фреймворка, но в то же время рассматривать Xamarin как фреймворк для написания приложения с нуля можно, если вы уверены, что сможете без особых проблем мигрировать приложение с Xamarin в MAUI, когда выйдет стабильная версия MAUI.

Вывод

Таким образом, все кроссплатформенные фреймворки реализуют сложные архитектуры для того, чтобы использовать общую кодовую базу для нескольких платформ, но при этом предоставлять доступ к нативным библиотекам и интерфейсам, таким как: взаимодействие с сетью, контакты, файловая система, уведомления и так далее. Из проведенного исследования можно заключить, что кроссплатформенные фреймворки реализуют гибкую архитектуру для того, чтобы поддерживать различные платформы и добавлять поддержку новых при необходимости. В то же время архитектура ориентирована на достижение лучшей производительности решения и удобства разработки.

Список литературы

1. Flutter architectural overview [Электронный ресурс]. – Режим доступа: https://docs.flutter.dev/resources/architectural-overview
2. Что такое Xamarin [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/xamarin/get-started/what-is-xamarin
3. .NET Multi-platform App UI documentation [Электронный ресурс]. – Режим доступа: https://docs.microsoft.com/ru-ru/dotnet/maui/
4. Обзор архитектуры React Native [Электронный ресурс]. – Режим доступа: https://reactnative.dev/docs/architecture-overview
5. Библиотека React Native Skia [Электронный ресурс]. – Режим доступа: https://github.com/react-native-skia/react-native-skia

6. Библиотека React Native Multithreading [Электронный ресурс]. – Режим доступа: https://github.com/mrousavy/react-native-multithreading
7. Kotlin Multiplatform [Электронный ресурс]. – Режим доступа: https://kotlinlang.org/docs/multiplatform.html
8. Kotlin Multiplatform Mobile [Электронный ресурс]. – Режим доступа: https://kotlinlang.org/lp/mobile/
9. Compose Multiplatform [Электронный ресурс]. – Режим доступа: https://www.jetbrains.com/lp/compose-mpp/

References

1. Flutter architectural overview [Electronic resource]. – Access mode: https://docs.flutter.dev/resources/architectural-overview
2. What is Xamarin [Electronic resource]. – Access Mode: https://docs.microsoft.com/en-us/xamarin/get-started/what-is-xamarin
3. .NET Multi-platform App UI documentation [Electronic resource]. – Access Mode: https://docs.microsoft.com/en-us/dotnet/maui/
4. Overview of React Native architecture [Electronic resource]. - Access mode: https://reactnative.dev/docs/architecture-overview
5. Library React Native Skia [Electronic resource]. – Access mode: https://github.com/react-native-skia/react-native-skia
6. Library React Native Multithreading [Electronic resource]. - Access mode: https://github.com/mrousavy/react-native-multithreading
7. Kotlin Multiplatform [Electronic resource]. – Access Mode: https://kotlinlang.org/docs/multiplatform.html
8. Kotlin Multiplatform Mobile [Electronic resource]. – Access Mode: https://kotlinlang.org/lp/mobile/
9. Compose Multiplatform [Electronic resource]. – Access Mode: https://www.jetbrains.com/lp/compose-mpp/