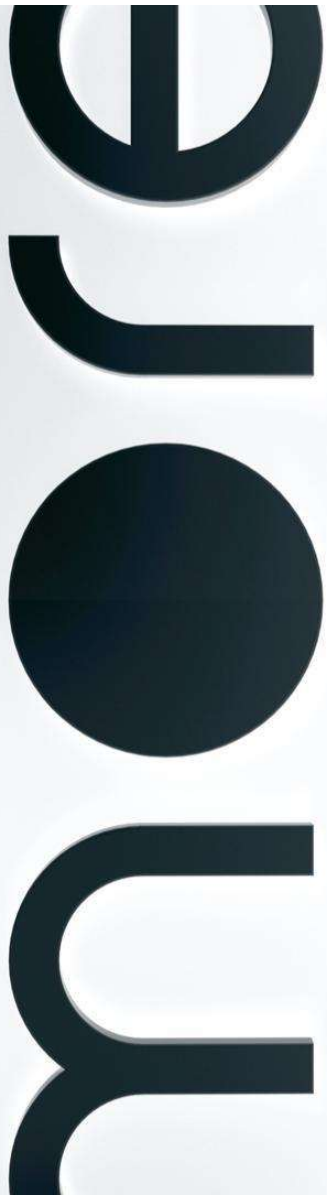
Александр Усков

Автоматное программирование и его применение в видеостриминге

VideoTech23



Показатели видеоплатформы за 2022 год:

>165 млн
уникальных пользователей

>500 пб
видеоконтента, отданного пользователям

>1,2 млрд
сессий смотрения

>1,6 млрд
рекламных показов

Спектр устройств

~ 15 тыс.
различных
моделей

~ 2,5 тыс.
различных
версий браузеров

>100
версий операционных
систем

История плеерной платформы

v 2.33 (Q2 2019)

- Монолит на Ruby, релиз загрузкой файлов
- до 15% всех видеосессий завершаются ошибками
- 4 года легаси-кода, минимум тестов, отсутствие документации

История плеерной платформы

v 3.0 (Q2 2020)

- NodeJS-микросервис, Typescript
- фичеризация, VideoAds SDK, бизнес-аналитика
- телеметрия и регистрация ошибок
- error rate ~8%

История плеерной платформы

v 4.0 (Q1 2021)

- Фикс более 300 багов
- Пересмотр бизнес-логики
- Публичные API, A/B тесты, общий интерфейс для всех площадок
- error rate ~6%

История плеерной платформы

v 5.0 (Q4 2022)

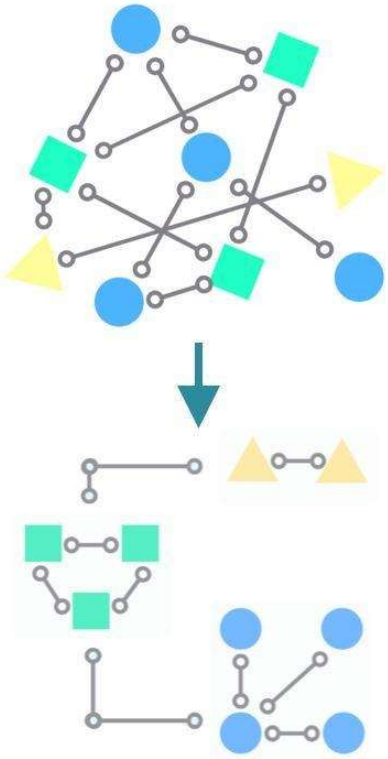
- Тотальный рефакторинг архитектуры
- Внедрение React
- Рост метрик смотрения при отсутствии дополнительного функционала
- error rate ~3%

Проблемы, которые наследуются

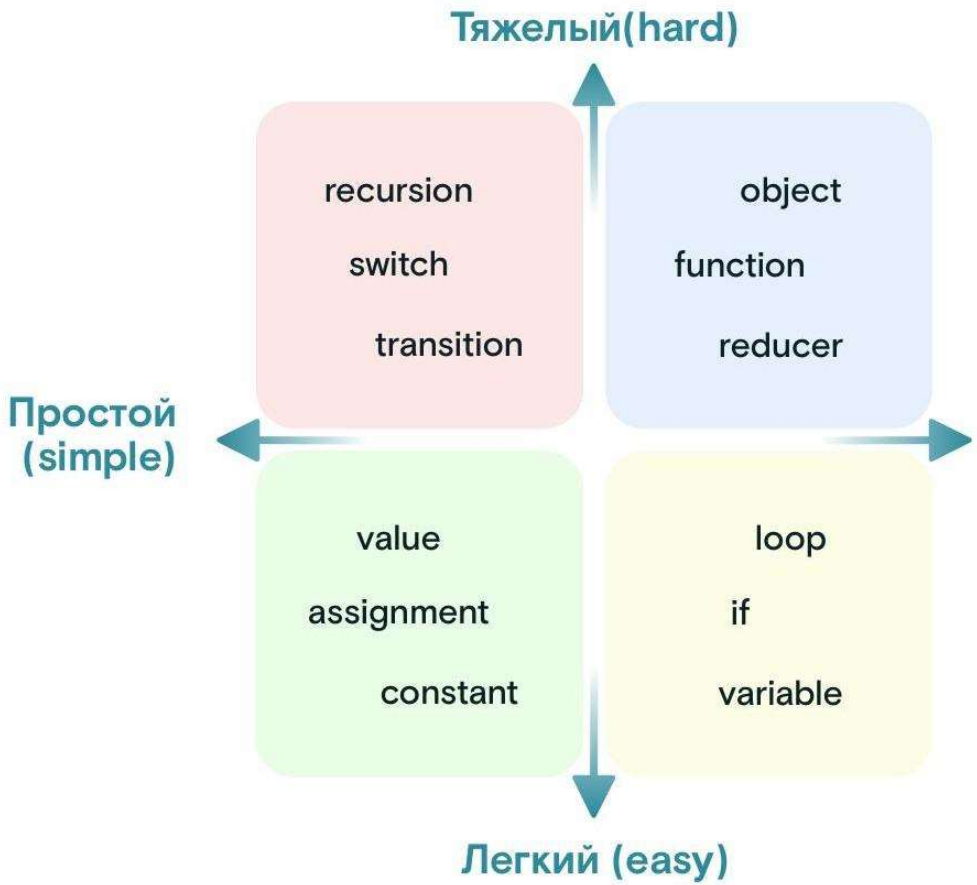
- Нет явных слоев в приложении (сетевой слой, бизнес-логика, view слой)
- Ванильный JS (позже TS), прямая работа с dom api
- Архитектура на основе pub/sub, Callback hell
- Нет внедрения зависимостей , прямые импорты и монолитизация кода

Автоматное программирование

- Декларативное описание бизнес-логики
- Low coupling / High Cohesion кодовой базы
- Синхронный код (преимущественно)



Сложность программных решений



Необходимая сложность

Случайная сложность

Модель состояния

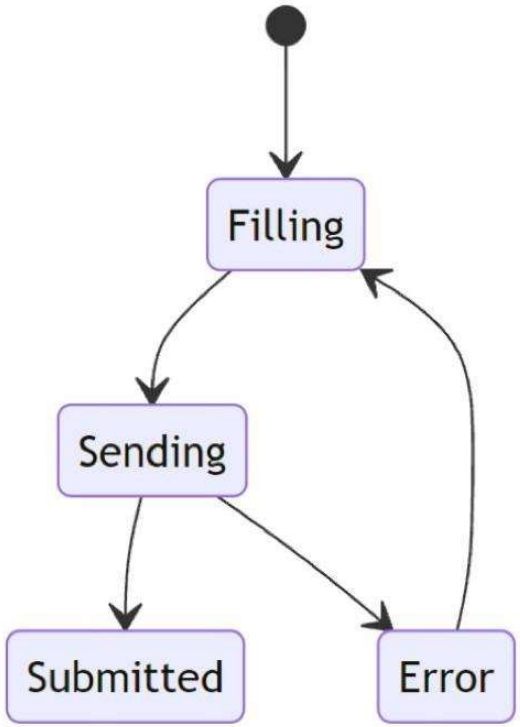
Управляющие состояния (control states)

- Описывают качественные отличия
- Определяют действия и процессы
- Количество ограниченно и растет линейно

Вычислительные состояния (computational states)

- Содержат количественные значения
- Определяют данные как результаты действий
- Количество велико и растет непредсказуемо

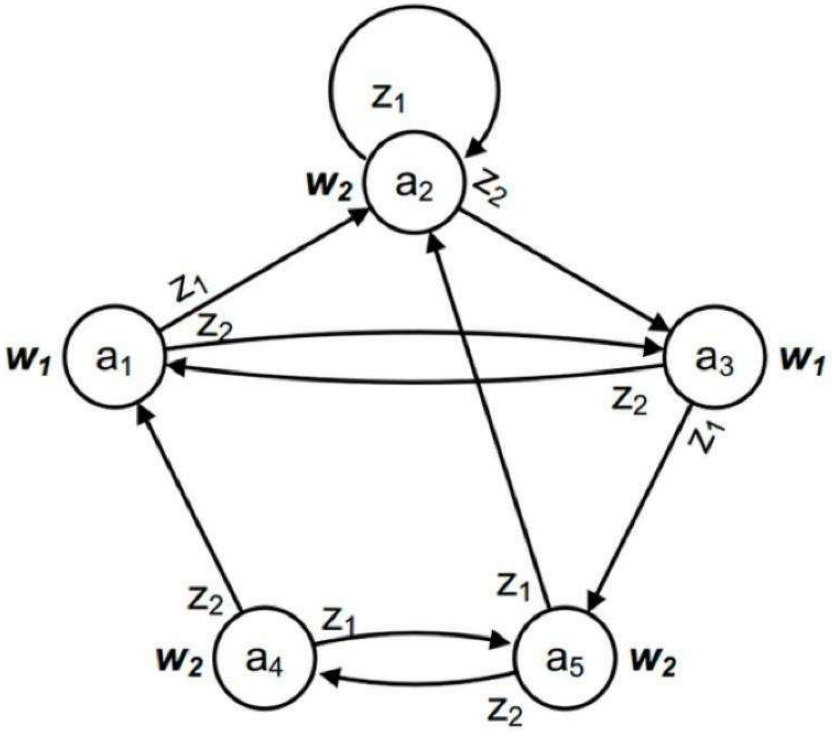
Проклятие флагов




```
const state = {
  registrationProcess: {
    valid: true,
    submitDisabled: true,
    inputDisabled: true,
    showSpinner: true,
    blockAuthentication: true,
  }
};
```

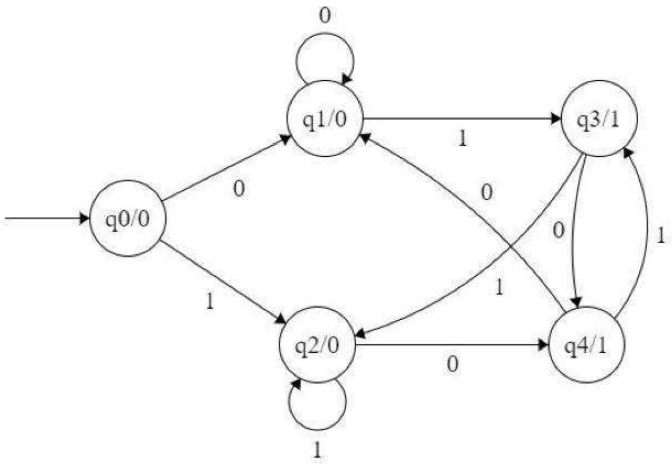
```
const state = {
  registrationProcess: {
    errors: ['name', 'address'],
    state: 'Error',
  }
};
```

Конечный автомат

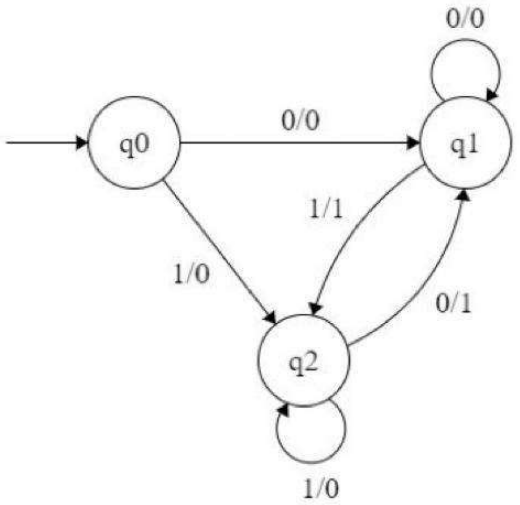


- Дискретный список состояний
- Дискретный список входных значений
- Дискретный список выходных значений
- Транзакционные детерминированные изменения

Машины Мили и Мура



Машина Мура: Следующее состояние является функцией текущего

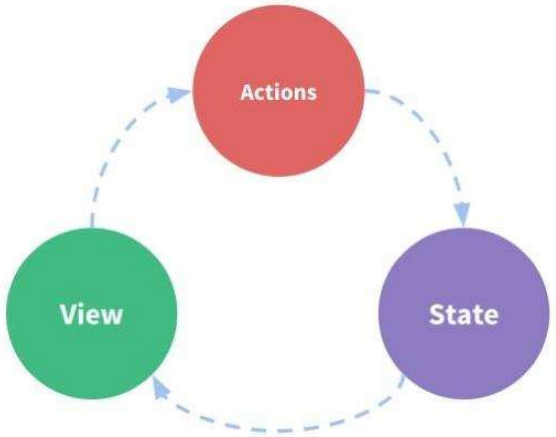


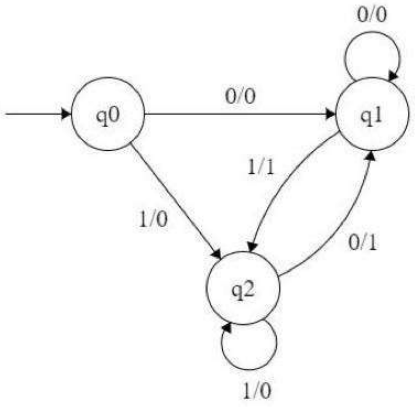
Машина Мили: Следующее состояние является функцией текущего состояния и входных значений

more.tv

Машина Мили vs Redux

- Состояние монолитно
- Глобальный контракт хранилища
- Модульная бизнес-логика (toolkit)
+ Можно превратить в автомат

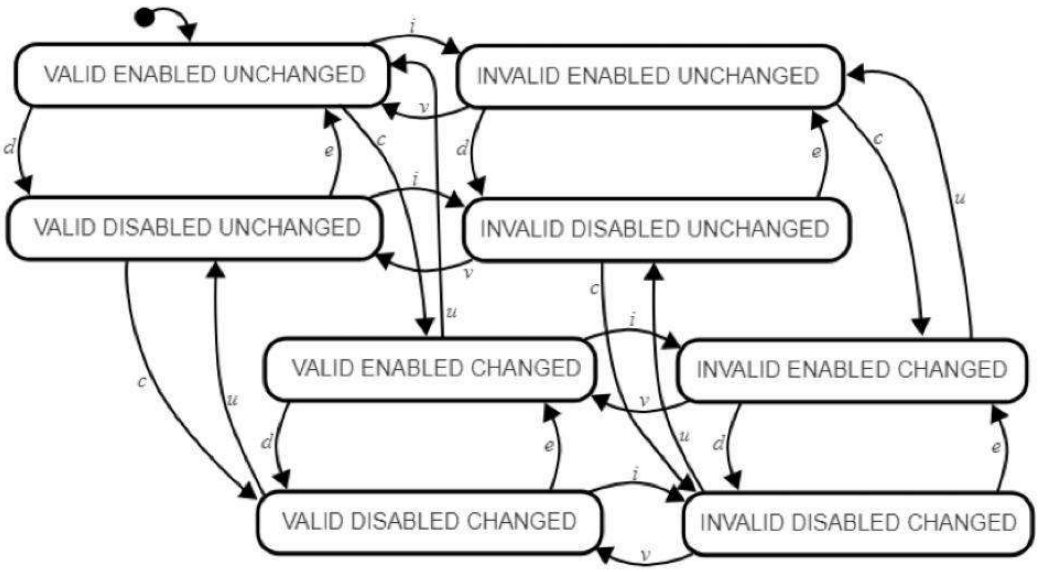




- Управляющие состояния являются контекстом вычислительных
- Хранилища изолированы и имеют динамический контракт
- Фабричная бизнес-логика

Взрыв состояний

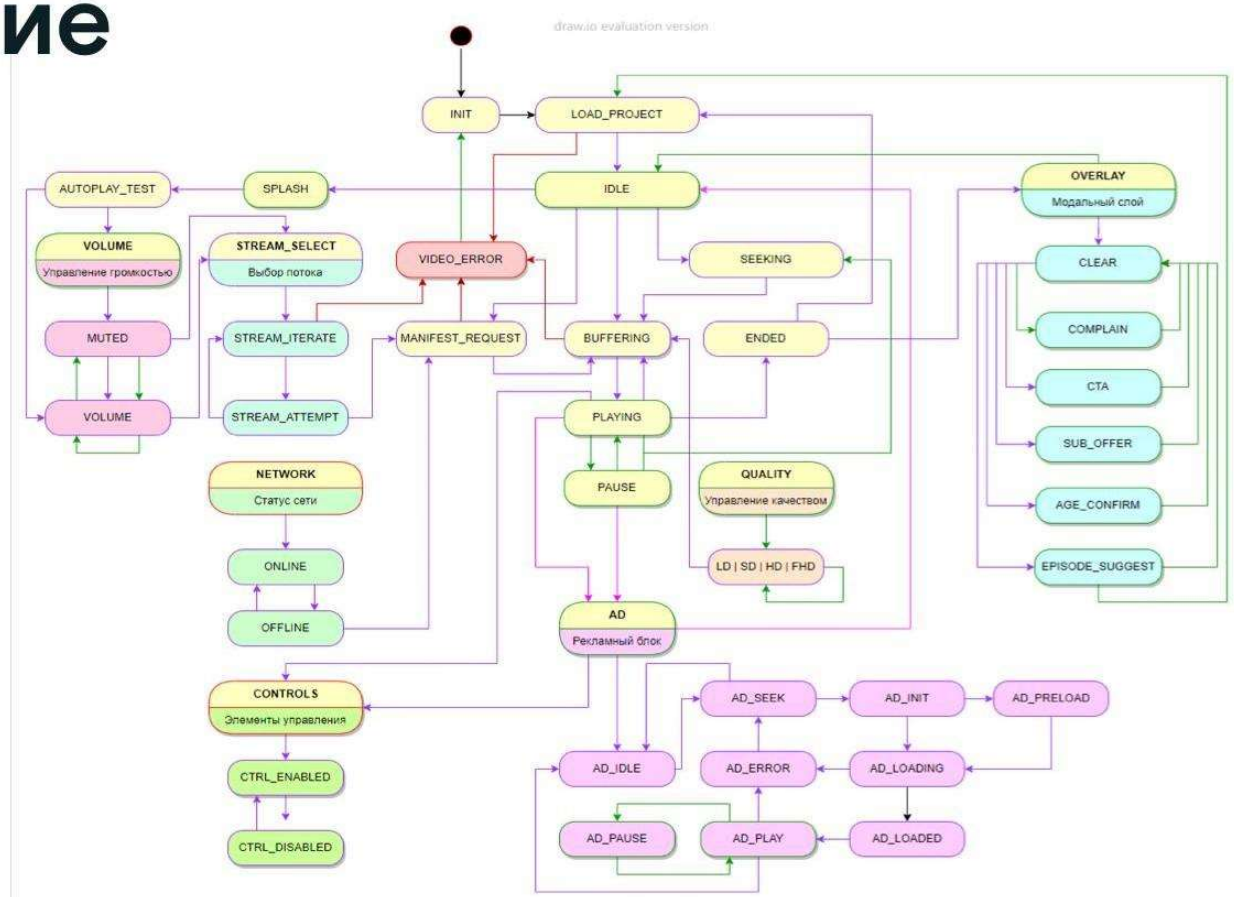
Обобщенное состояние приводит к слишком высокой сложности



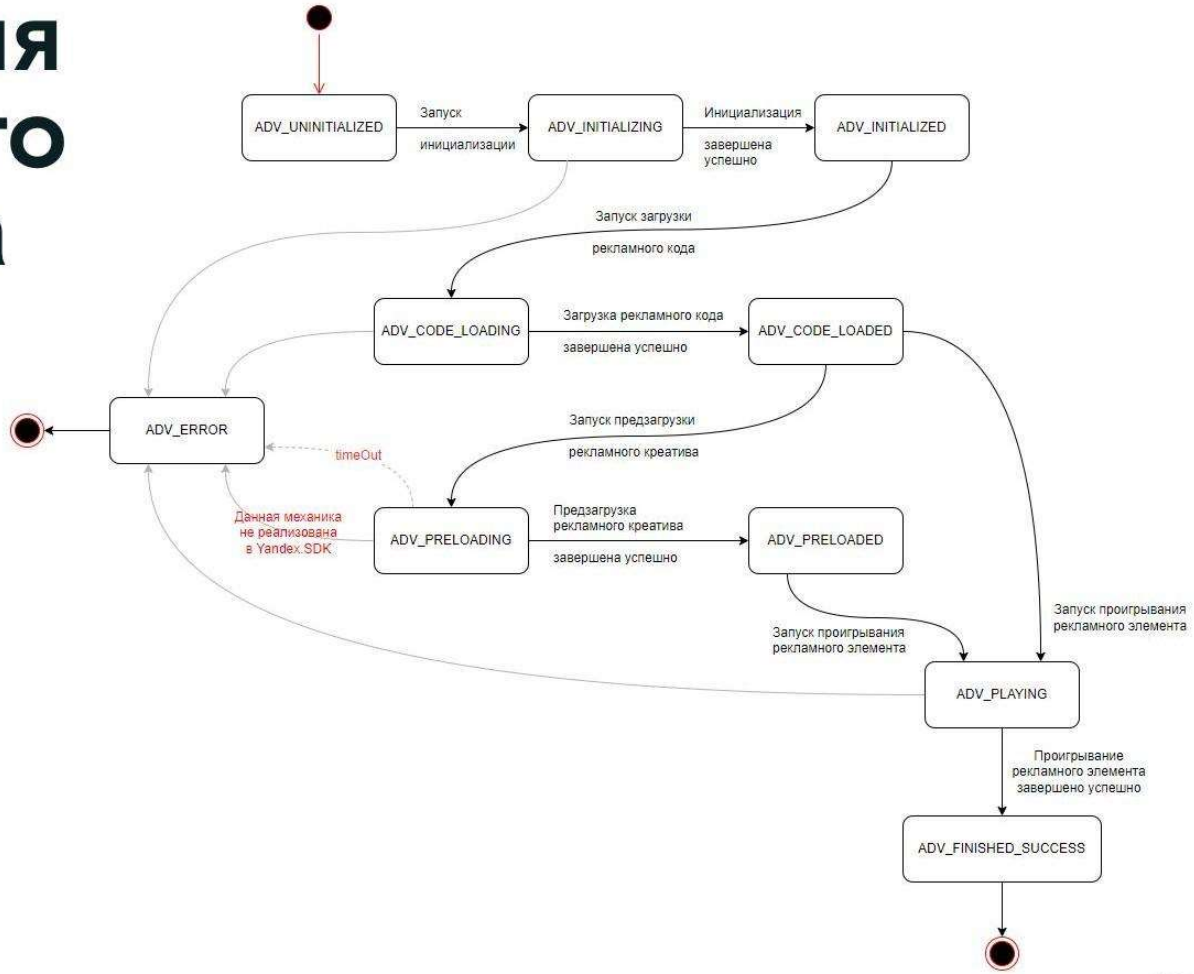
Решения:

- вложенные автоматы
- параллельные автоматы
- условные переходы (guards)

Проектирование управляющих состояний



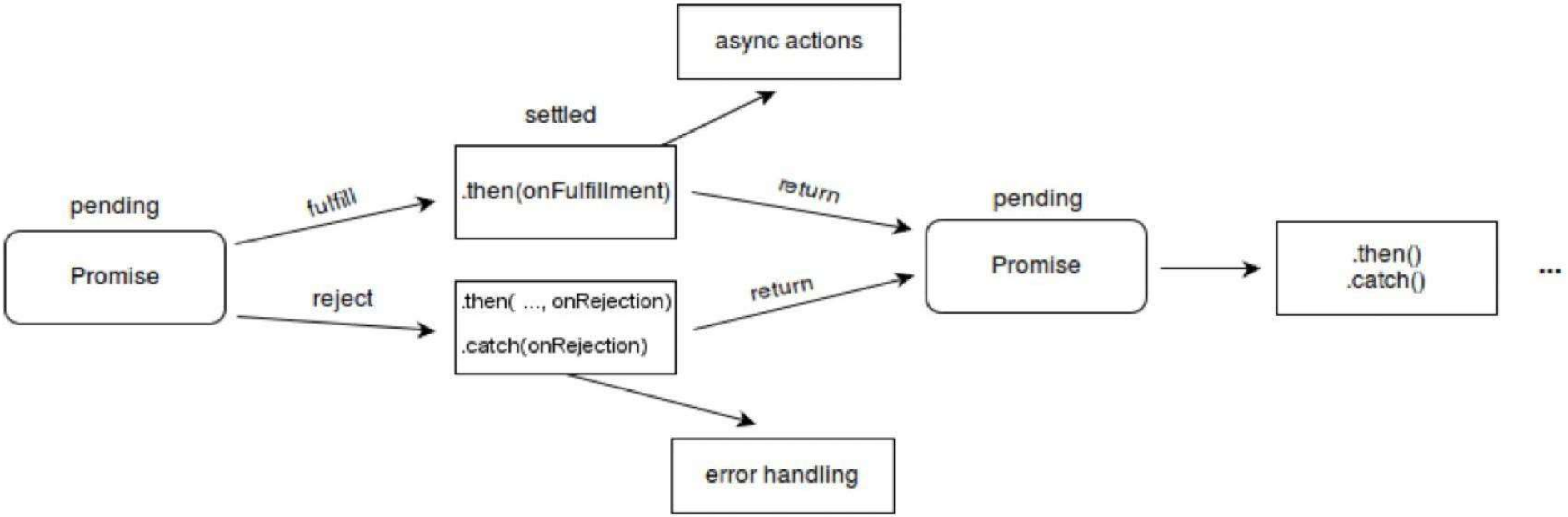
Реализация рекламного примитива



Управление состоянием

- Все управляющие состояния и события являются словарями (enum)
- Вычислительные состояния являются контекстом или производными от управляющих
- Новое состояние определяется текущим состоянием и событием

Promise как автомат



Синхронизация Promise

```
const preload = () ⇒ {
  if (_preload)
    return _preload;
  _preload = new Promise(
    async (resolve, reject) ⇒ {
      reject(1);
    });
  return _preload;
};
```

```
while (1)
  switch (
    (_context.prev = _context.next)
  ) {
    case 0:
      reject(1);
    case 1:
    case "end":
      return _context.stop();
  }
```

Родственные практики

- **Динамическая диспетчеризация**

  Перечислимые управляющие состояния можно мапить на любые обработчики

Родственные практики

- Динамическая диспетчеризация

  Перечислимые управляющие состояния можно мапить на любые обработчики

- **Чистые функции**

  Обработчики имеют минимум зависимостей и могут быть реализованы идемпотентно

Родственные практики

- Динамическая диспетчеризация

  Перечислимые управляющие состояния можно мапить на любые обработчики

- Чистые функции

  Обработчики имеют минимум зависимостей и могут быть реализованы идемпотентно

- **Инкапсуляция логики**

  Управляющие процессы описаны там же, где и управляющие состояния

Родственные практики

- Динамическая диспетчеризация

  Перечислимые управляющие состояния можно мапить на любые обработчики

- Чистые функции

  Обработчики имеют минимум зависимостей и могут быть реализованы идемпотентно

- Инкапсуляция логики

  Управляющие процессы описаны там же, где и управляющие состояния

- **Параметрический полиморфизм**

  Перечислимые типы позволяют применять исчерпывающие проверки значений

Решения на базе JS/TS

Redux-toolkit

+ Крупнейшее коммьюнити и база знаний
+ Лучшие инструменты отладки
+ Декларативная конфигурация
– Легко использовать неправильно:

```
dispatch(SomeSliceActions.setRefValue({
  // immutable object constructor
}));
```

Решения на базе JS/TS

XState

+ Самое развитое frontend-решение
+ Есть визуализация состояний и Live-редактор
+ Поддерживает вложенные автоматы и guards
– Сложная конфигурация
– Временами, слишком императивный код

Решения на базе JS/TS

JSSM

+ Использует DSL для описания автоматов
+ Есть live-редактор
+ Удобный декларативный код
– Относительно маленькое community
– Мало примеров за пределами официальной документации

Решения на базе JS/TS

Все прочее

+ Адаптировано для конкретных фреймворков
– Не обновляется годами

Решение на базе Redux/Toolkit

```
▶ updater (pin): { step: "READY" }
▶ playback (pin): { step: "PAUSED", currentTime: 19.746, duration: 1448.44, … }
▶ watchpoint (pin): { step: "READY", previousTime: 19, points: […] }
▶ heartbeat (pin): { step: "IDLE", previous: {…}, progress: {…}, … }
▶ rewind (pin): { step: "READY" }
▶ rewindAcc (pin): { step: "READY", type: "inc", inc: 0, … }
▼ buffering (pin)
    step (pin): "READY"
    startAt (pin): null
    bufferingTime (pin): 0
    initialBufferTime (pin): 0
    bufferedEnd (pin): 70
    loadedPercent (pin): 4.832785617630002
▶ adTimeNotify (pin): { step: "DISABLED", points: [], time: null }
```

Решение на базе Redux/Toolkit

+ Все автоматы – это один глобальный редуктор, что позволяет легкую отладку через Redux Devtools

Решение на базе Redux/Toolkit

+ Все автоматы - это один глобальный редуктор, что позволяет легкую отладку через Redux Devtools

+ **Конфиг автомата - максимально плоский, в нем описывается только матрица переходов. Guards и прочая комбинаторика описывается в reducers**

Решение на базе Redux/Toolkit

+ Все автоматы – это один глобальный редуктор, что позволяет легкую отладку через Redux Devtools

+ Конфиг автомата – максимально плоский, в нем описывается только матрица переходов. Guards и прочая комбинаторика описывается в reducers

+ **Логика без побочных эффектов в reducerax, остальное в effectax, гдечерез зависимости есть доступ ксервисному слою**

Решение на базе Redux/Toolkit

+ Все автоматы - это один глобальный редуктор, что позволяет легкую отладку через Redux Devtools

+ Конфиг автомата - максимально плоский, в нем описывается только матрица переходов. Guards и прочая комбинаторика описывается в reducers

+ Логика без побочных эффектов в reducerax, остальное в effectax, гдечерез зависимости есть доступ ксервисному слою

+ **React обеспечивает реактивность минимальными усилиями, при этом компоненты «глупые» и детерминистически отражают состояние**

Решение на базе Redux/Toolkit

+ Все автоматы - это один глобальный редуктор, что позволяет легкую отладку через Redux Devtools

+ Конфиг автомата - максимально плоский, в нем описывается только матрица переходов. Guards и прочая комбинаторика описывается в reducers

+ Логика без побочных эффектов в reducerax, остальное в effectax, гдечерез зависимости есть доступ ксервисному слою

+ React обеспечивает реактивность минимальными усилиями, при этом компоненты «глупые» и детерминистически отражают состояние

– **Нет возможности отрисовать весь граф состояний разом, только отдельные автоматы**

Решение на базе Redux/Toolkit

+ Все автоматы - это один глобальный редуктор, что позволяет легкую отладку через Redux Devtools

+ Конфиг автомата - максимально плоский, в нем описывается только матрица переходов. Guards и прочая комбинаторика описывается в reducers

+ Логика без побочных эффектов в reducerax, остальное в effectax, гдечерез зависимости есть доступ ксервисному слою

+ React обеспечивает реактивность минимальными усилиями, при этом компоненты «глупые» и детерминистически отражают состояние

– Нет возможности отрисовать весь граф состояний разом, только отдельные автоматы

– **Эффекты автомата должны содержать минимум асинхронного кода за раз, такие эффекты легче «отменить»**

Standalone + React Hooks

```
createMachine({
  config: {···
  },
  initialState: 'IDLE',
  initialContext: {},
  // optional
  reducer: ({ state, context }, { type, payload }, config) => {···
  },
  effects: {···
  },
});
```

Конфигурация автомата

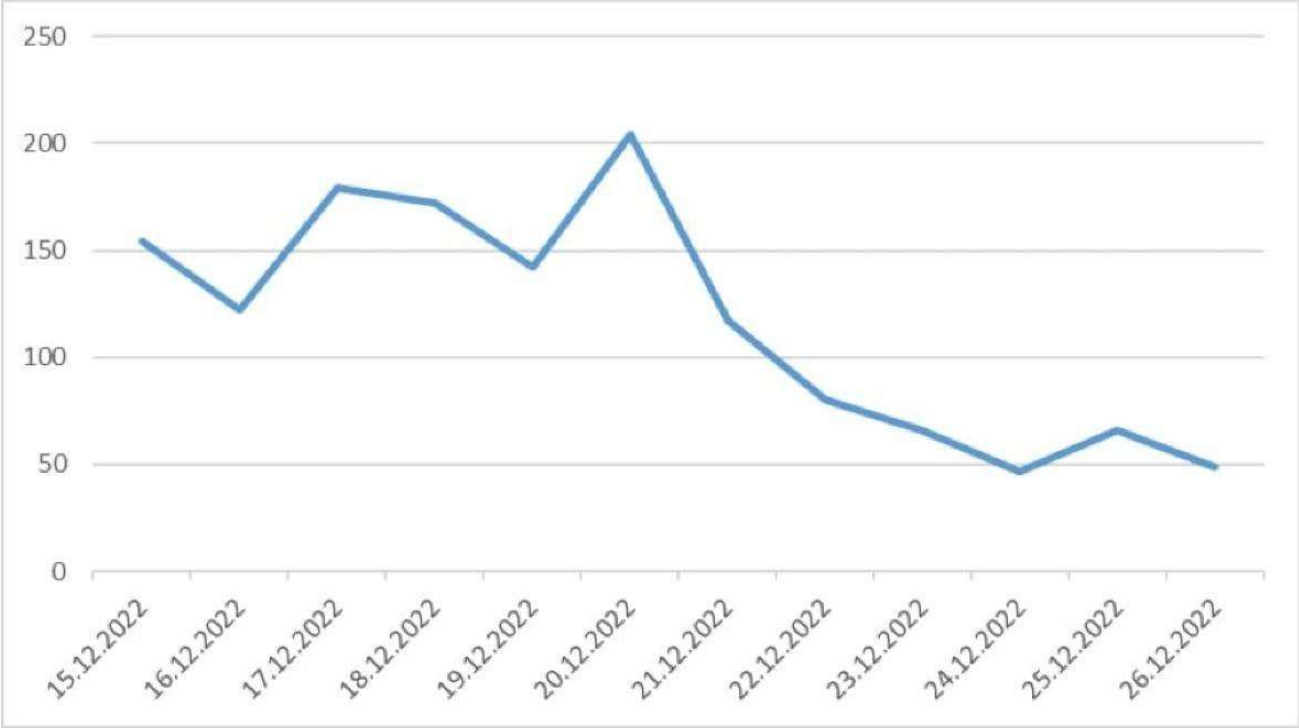
```
config: {
  IDLE: {
    INIT_P2P: 'INITIALIZE_P2P_PENDING',
  },
  INITIALIZE_P2P_PENDING: {
    INITIALIZE_P2P_RESOLVE: 'INITIALIZED',
    INITIALIZE_P2P_REJECT: 'IDLE',
  },
  INITIALIZED: {
    CHANGE_TRACK: 'DISPOSING_P2P',
    AD_BREAK_STARTED: 'DISPOSING_P2P',
    INIT_RESUME_VIDEO: 'IDLE',
  },
  DISPOSING_P2P: {
    DISPOSING_P2P_RESOLVE: 'IDLE',
  },
},
```

Архитектура автоматного приложения

- MVC, MVVM, ECS – имеет значение только инкапсуляция состояния
- Автомат реализует жизненный цикл модели/сущности
- Рендер происходит асинхронно на основе состояния («реактивно») и независим от контекста
- Единая шина событий синхронизирует все автоматы

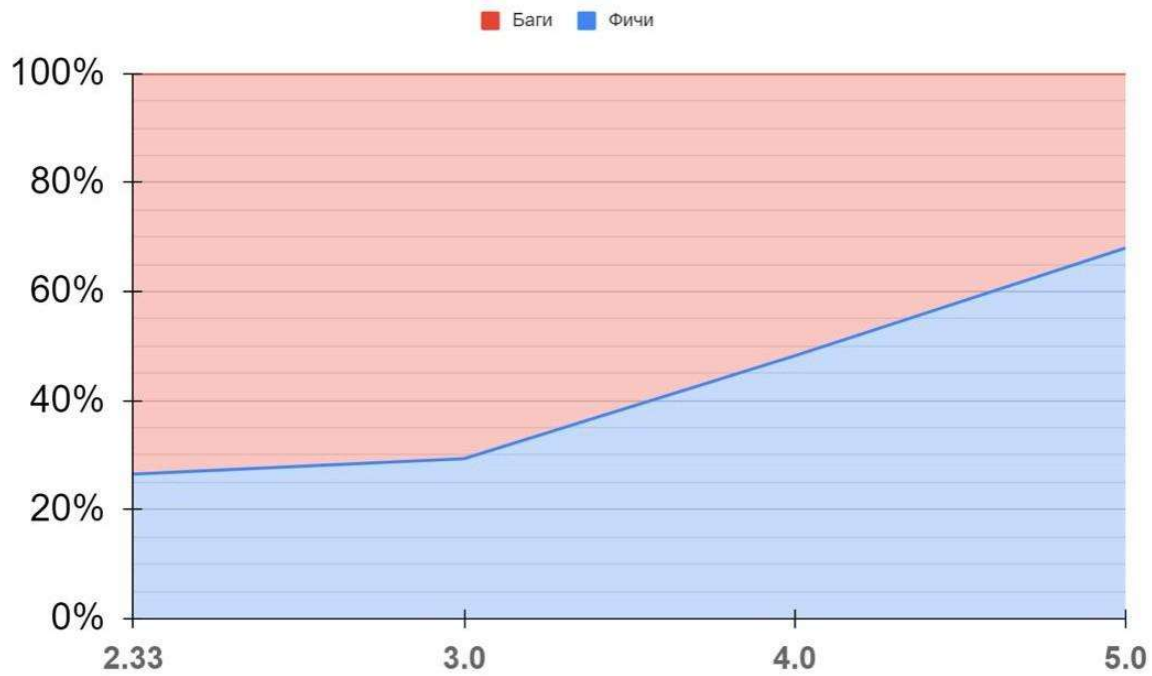
Результаты релиза

Обращения в поддержку



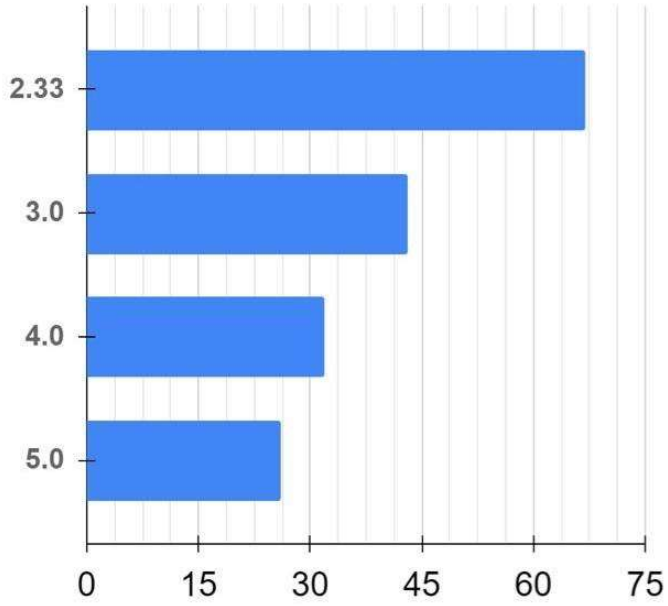
Результаты релиза

Состав версий

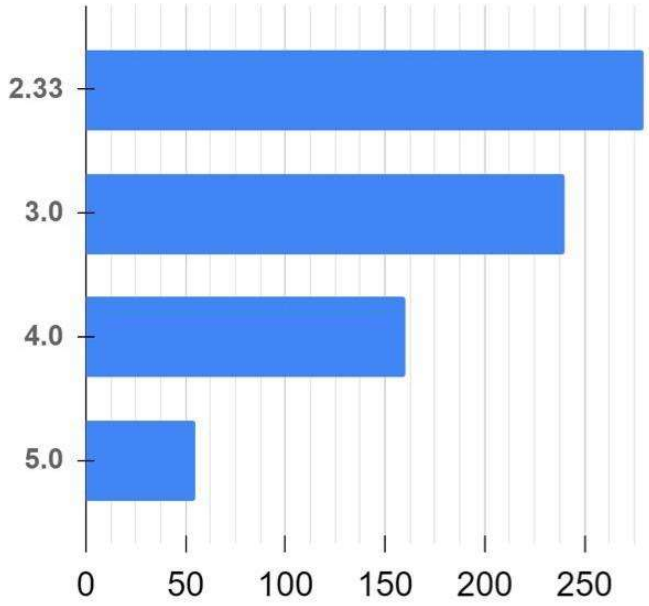


Результаты релиза

TTM (дни)



Репорты пользователей



Автоматный backend

~~REST~~ RPC

- gRPC, ODBC, NFS, CORBA, WAMP и др.
- Анемичная модель данных
- CQRS, at-most-once, pub-sub
- «Serverless» обработчики + полутонкие клиенты

Documentation-As-Code

MermaidJS

Клиентская библиотека
для отображения диаграмм
с синтаксисом, близким к Markdown

SCXML

State Machine Notation for Control
Abstraction

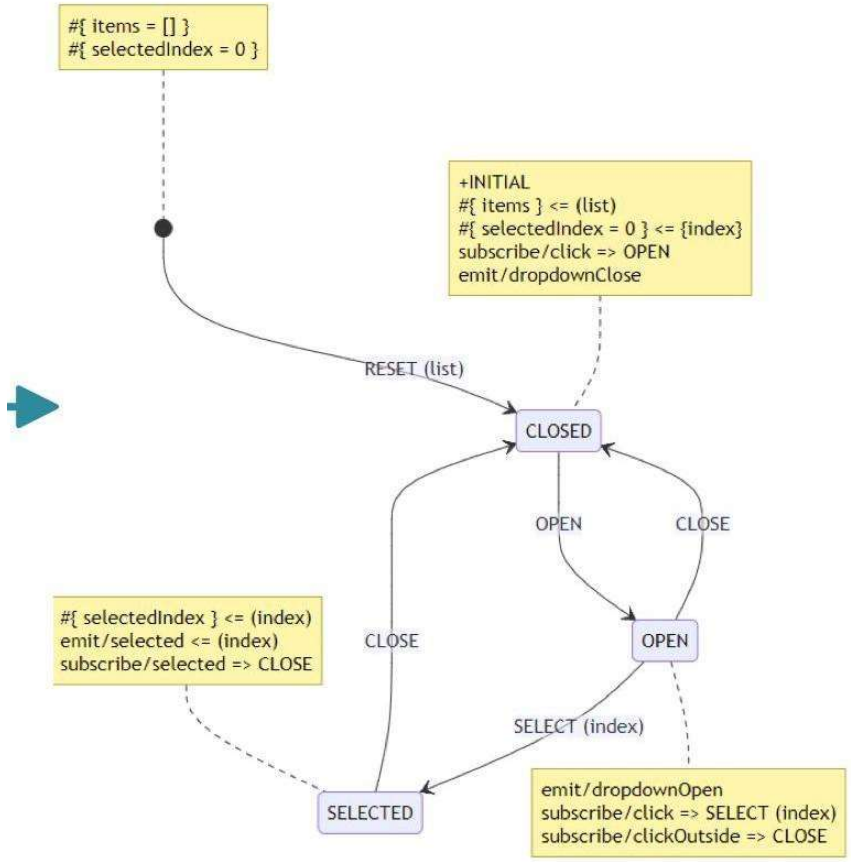
PlantUML

DSL+Набор инструментов для диаграмм
и визуальных спецификаций

- Визуальный редактор
- Интеграция GitHub/GitLab
- Готовые парсеры
- Генерация кода на основе спецификации

DSL и метапрограммирование

```
[*] ⟶ CLOSED: RESET (list)
note left of [*]
#{ items = []}
#{ selectedIndex = 0 }
end note
CLOSED ⟶ OPEN: OPEN
OPEN ⟶ CLOSED: CLOSE
OPEN ⟶ SELECTED: SELECT (index)
SELECTED ⟶ CLOSED: CLOSE
note left of CLOSED
+INITIAL
#{ items } ← (list)
#{ selectedIndex = 0 } ← {index}
subscribe/click ⇒ OPEN
emit/dropdownClose
end note
note left of SELECTED
#{ selectedIndex } ← (index)
emit/selected ← (index)
subscribe/selected ⇒ CLOSE
end note
note right of OPEN
emit/dropdownOpen
subscribe/click ⇒ SELECT (index)
subscribe/clickOutside ⇒ CLOSE
end note
```



Раздаточные материалы