

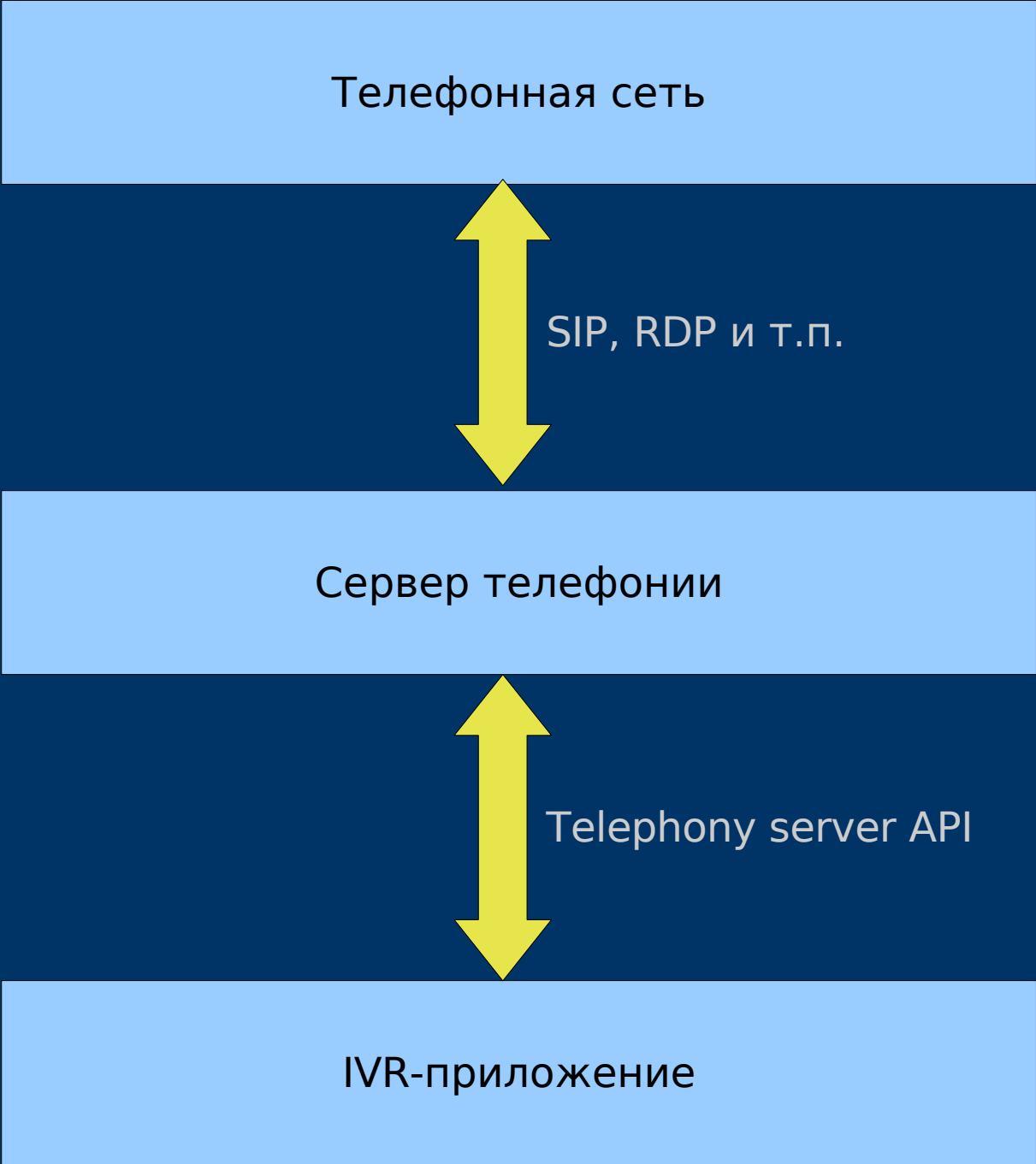
# *Программирование IVR-сервисов*

Или, как я перестал бояться и  
полюбил javascript

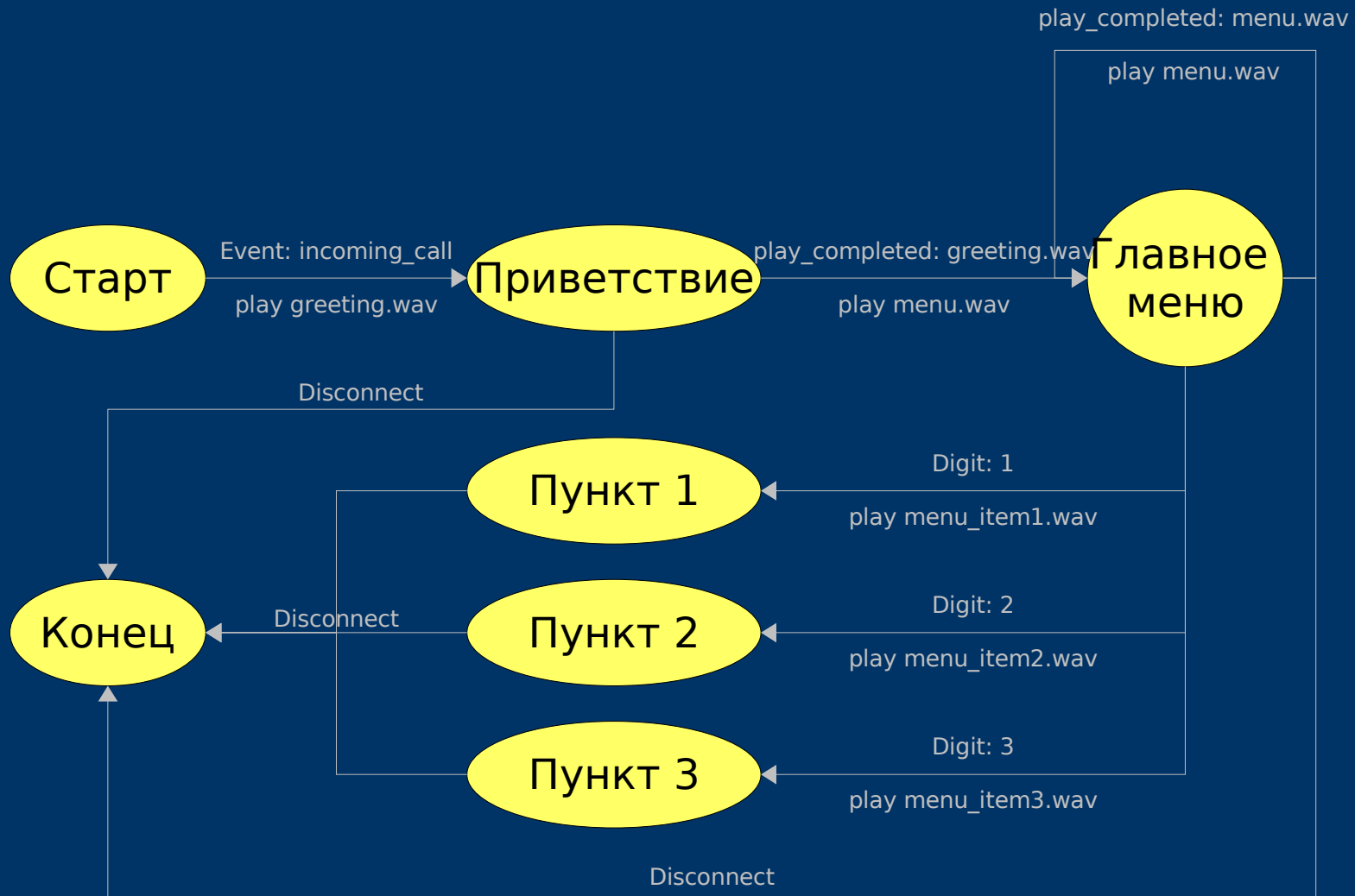
Алексей Григорович  
Shamrock technologies  
RuPy.Ru '08

---

---



# Схема состояний



*Как это программировать?*



# Примитивный подход

```
function Run(appctx) {
    var event;
    var state = STATE_START;
    while(event = appctx.getEvent()) {
        if (event.Type == 'IncomingCall'
            && state == STATE_START)
        {
            appctx.Answer();
            var reqId = appctx.PlayFile('greetings.wav');
            state = STATE_GREETINGS;
        } else if (state == STATE_GREETINGS
                    && event.Type == 'PlayCompleted'
                    && event.ReqID == reqId)
        {
            state = STATE_MAIN_MENU;
            reqId = appctx.PlayFile('main_menu.wav');
        } // else if (...)
    }
}
```

---

---

# *Очевидно, это не сработало*

- не масштабируется
  - каждое новое состояние потенциально затрагивает все уже имеющиеся
  - трудно добавлять новый функционал
  - куча трудновоспроизводимых ошибок

# *Что делать?*

- Пришлось внимательнее посмотреть на возможности языка
- Был приятно удивлен!



# *Откровение первое*

Оказывается, есть настоящая спецификация!



# *Откровение первое*

OMG I CAN HAS A REAL SPEC!!1



# Откровение второе

## Настоящие замыкания

```
// compose(f, g)(x) = f(g(x))  
function compose(f, g) {  
  return function(x) {  
    return f(g(x));  
  }  
}
```

# Откровение второе

## Настоящие замыкания

```
function withdraw(acct_from, acct_to, amount) {  
  Database.connect(function(c) {  
    Database.transaction(function() {  
      c.execute("UPDATE accounts SET balance = balance - ?  
                WHERE id = ?", amount, acct_from.id);  
      c.execute("UPDATE accounts SET balance = balance + ?  
                WHERE id = ?", amount, acct_to.id);  
    });  
  });  
}
```

---

---

# Откровение третье

Настоящая объектная система

несмотря на простой вид:

```
var quiz_question = {  
  text: "How many people will sit through the whole talk?",  
  
  correct_answer: 1,  
  
  check_answer: function(answer) {  
    return answer == this.correct_answer;  
  }  
}
```

# Откровение третье

Настоящая объектная система

ОО там (почти) до самого низа:

```
(1).toString();  
// => "1"
```

```
[1, 2, 3].slice(1);  
// => [2, 3]
```

# Откровение третье

Настоящая объектная система

ОО там (почти) до самого низа:

```
Number.prototype.incr = function() { return this + 1 };  
Number.prototype.decr = function() { return this - 1};  
(1).incr() // => 2  
(1).incr().decr() // => 3
```

# Откровение третье

## Настоящая объектная система

```
/* f.if_only(g) when applied to 'x' returns a function that calls
 * 'f(x)' if only g(x) has returned something meaningful.
 */
Function.prototype.if_only = function(g) {
  var f = this;
  return function() {
    var ret = g.apply(this, arguments);
    if (ret) {
      ret = f.apply(this, arguments);
    }
    return ret;
  }
}
```

---

---

# *Результат*



# Результат

```
var MenuController = {
  enter: function() { return this.greetings(); },
  greetings: function() {
    return new State('greetings', function() {
      this.jump_after_play(MenuController.menu(),
        'greetings.wav');
    });
  },
  menu: function() {
    return new State('main_menu', function() {
      this.jump_after_play(MenuController.menu(), 'menu.wav');
      this.jump_on_digit(MenuController.item(1), 1);
      this.jump_on_digit(MenuController.item(2), 2);
      this.jump_on_digit(MenuController.item(3), 3);
    });
  },
  item: function(item_no) { /* ... */ }
}
```

---

---

# Результат

```
Driver.prototype = {
  run: function(context) {
    try {
      var state = start_state;
      state.enter();
      while(state !== this.state_end) {
        state = state.handle(context.getEvent());
      }
    } catch(e) {
      this.logger.error('exception in state ' + state,
e);
    } finally {
      this.on_disconnect(state);
    }
  }
}
```

# Результат

```
function State(name, enterproc) {  
  this.name = name;  
  this.enter = enterproc;  
}
```

```
State.prototype = {  
  /**  
   * Set up current state. Called when entering this state.  
   */  
  enter: function() { },  
  
  /**  
   * handle :: event -> (State | null)  
   * Processes an event and returns a new state, or null  
   * if event is ignored.  
   */  
  handle: function(event) { }  
}
```

---

---

# Результат

```
var StateActions = {
  /*
   * plays <thing> and installs a handler that jumps to <state>
   * after it has finished playing.
   */
  jump_after_play: function(state, thing) {
    var handler = this.jump_to(state);
    return this.handle_after_play(handler, thing);
  },

  handle_after_play: function(handler, thing) {
    var matcher = this.play(thing);
    if (!matcher) {
      throw "handle_after_play: couldn't play: " + thing;
    }
    return this.override(handler.if_only(matcher));
  }
}
```

---

---

# Уроки

- Функциональный подход
    - всю информацию, необходимую новому состоянию, передавать в аргументах конструктору
    - Изменчивые (mutable) объекты-состояния оказались источником ошибок номер один
    - Даже такие простые вещи, как сохранение индекса текущего элемента в `this` частенько кусались.
- 
-

# Уроки

- Расширяемый динамический язык
    - легко заточить под свои нужды
    - в том числе, можно легко расширять стандартную библиотеку
    - ... что стоит делать если только весь контроль над исполняемым кодом находится у вас
- 
-

# Уроки

- Плюсы javascript:
    - Симпатичная смесь функционального/ОО подходов
    - Почти любой встроенный объект можно поменять так, как тебе надо
    - можно написать «стандартную библиотеку» -- такую, как тебе надо
- 
-

# Уроки

- Минусы javascript
  - местами недостаточно гибкий
  - **придется** писать «стандартную библиотеку» -- такую, как тебе надо

# Уроки

- Минусы javascript
    - местами недостаточно гибкий
    - **придется** писать «стандартную библиотеку» -- такую, как тебе надо
    - включая module system
- 
-

***Вопросы?***

