

Санкт-Петербургский государственный электротехнический университет  
«ЛЭТИ» им. В.И. Ульянова (Ленина) (СПбГЭТУ)

Работа допущена к защите  
зав. кафедрой

\_\_\_\_\_ Советов Борис Яковлевич

«\_\_\_\_» \_\_\_\_\_ 2012 г.

## ВЫПУСКНАЯ РАБОТА БАКАЛАВРА

Тема: **Использование динамических библиотек в рамках Flash-приложений**

Направление: 230400.62 – Информационные системы

Выполнил студент гр. 8372 \_\_\_\_\_ Борисенко Константин Алексеевич

Научный руководитель,  
асс. каф. АСОиУ СПбГЭТУ \_\_\_\_\_ Степуленок Денис Олегович

## **Аннотация**

В данной дипломной работе разработан программный продукт, представляющий собой web-сервер на базе Node.js, который позволяет вызывать функции, реализованные в динамической библиотеке из Flash-приложений или других технологий (например, Javascript, Silverlight) для создания клиентской части web-приложения.

В дипломной работе разработана общая схема системы, проанализированы достоинства и недостатки возможных средств реализации, и выбрано наилучшее с точки зрения сформулированных критериев.

С целью сравнения приведены листинги web-серверов, реализованных на разных языках (C, Python, PHP, Node.js).

Разработан исходный код сервера на Node.js, описан процесс настройки Node.js под Linux.

Эту работу можно использовать для дальнейшей разработки задач КИО на Flash, требующих использование функций из динамических библиотек.

В заключении описаны возможные перспективы программного продукта. Большая часть поставленных в техническом задании целей была достигнута.

## **Abstract**

In this diploma has been created a software, which is a server made on Node.js, which allows to connect dynamic libraries with Flash-applications or other technologies (ex. Javascript, SilverLight) for making client parts of web-services.

The common structure of the client has been shown, advantages and disadvantages of possible platforms has been analyzed, and the best solution according to formulated criteria has been chosen.

In order to compare listings of web-servers, made on C, Python, PHP, Node.js has been demonstrated.

The project has been made on Node.js. Process of setting-up Node.js on Linux has been shown.

This project can be useful for the future development of the competition CEO (to Construct, to Explore, To optimize) on Flash.

In the end there are future possibilities of this product. Allmost all of the goals, described in specification, has been reached.

## **Ключевые слова**

- Node.JS
- Javascript
- Создание, вызов динамической библиотеки
- Flash
- ActionScript
- Flash player debugger
- npm
- node-ffi

# Содержание

<b>Глава 1. Системный анализ предметной области и постановка задачи</b>	7
1.1. Предметная область	7
1.2. Постановка задачи дипломного проекта	13
<b>Глава 2. Используемые технологии</b>	15
2.1. Структура проекта	15
2.2. Выбор платформы для Flash	16
2.3. Выбор платформы для создания сервера	16
2.3.1. Протокол передачи	16
2.3.2. Среда создания	17
2.3.3. Создание тестовой динамической библиотеки	27
2.4. Структура проекта	29
<b>Глава 3. Проектирование</b>	30
3.1. Настройка Node.js	30
3.1.1. Настройка под Linux	30
3.1.2. Настройка под Windows	31
<b>Глава 4. Реализация программы</b>	38
4.1. Предварительная работа с динамической библиотекой	38
4.2. Оформление запроса	39
4.3. Структура проекта	40
4.3.1. Серверная часть	40
4.3.2. Клиентская часть	45
<b>Глава 5. Установка сервера участником</b>	49

<b>Глава 6. Заключение . . . . .</b>	50
6.1. Достигнутые результаты . . . . .	50
6.2. Дальнейшее развитие проекта . . . . .	51
<b>Глава 7. Используемая литература . . . . .</b>	52

# Глава 1

## Системный анализ предметной области и постановка задачи

### 1.1. Предметная область

Существует конкурс КИО - «Конструируй, Исследуй, Оптимизирай» (рис. 1.1). Он был создан 9 лет назад. Данный конкурс проводится для учеников с 1 по 11 класс. Существует три уровня:

- начальный уровень - для учеников начальных классов,
- уровень I - для учеников от 5 до 7 класса включительно,
- уровень II - для старших участников.

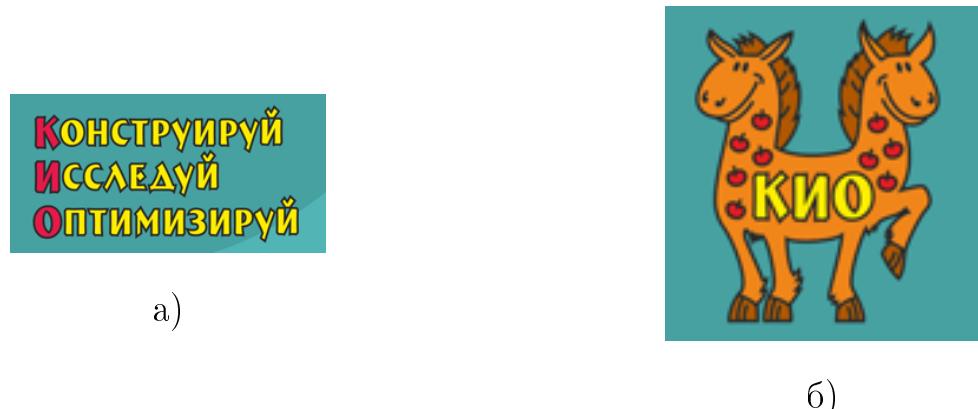


Рис. 1.1. Эмблемы конкурса КИО

Участникам каждого уровня предлагаются три логические задачи. Все задачи отражают недавно возникшее направление научных исследований, которое, применительно к разным наукам, получило названия компьютерной математики, компьютерной физики, компьютерной биологии и т. д. За каждым игровым сюжетом стоит серьёзная и, возможно, ещё не решённая в об-

щем виде задача, которая после окончания Конкурса подробно и популярно обсуждается на страницах журнала «Компьютерные инструменты в школе». Задания конкурса представлены в форме компьютерных моделей-лабораторий с игровыми элементами. В процессе работы с заданием участник конструирует частичные решения задачи, которые оцениваются по установленным в задании критериям. Таким способом формируется «рекорд» - число или набор чисел, характеризующие степень достижения поставленных в задании целей. Механизм рекорда позволяет участникам оценивать собственное продвижение в решении задач, а для жюри конкурса рекорды являются основой для составления рейтинга. Конкурс позволяет ребятам из самых отдаленных уголков России участвовать наравне с ребятами из больших городов, если у них есть Интернет или электронная почта. По окончании Конкурса каждый участник получает электронный сертификат, заверенный электронной подписью. Это означает, что выдаваемый жюри Конкурса электронный сертификат зашифрован так, что его можно проверить на подлинность открытием в программе «Электронный сертификат» на сайте Конкурса, но подделать его невозможно. Я участвовал в создании Конкурса КИО-2010. Я создал логическую задачу "Прожорливый тьюрмит". Для меня это был бесценный опыт, и я с радостью участвую в модернизации Конкурса сейчас. Созданная мною игра будет описана в примерах задач. Далее приведены примеры задач.

**Прожорливый тьюрмит (рис. 1.2)** Существо, которое в теории алгоритмов называют тьюрмитом (Тьюринг+термит), перемещается по тору, развертка которого представлена в виде клетчатого поля 20x20. Тор, который мы можем представить в виде бублика, получится из этого поля, если его «свернуть в трубочку», а затем соединить края трубы. При этом соседними станут верхние и нижние клетки каждого столбца и каждой строки исходного поля.

Управляетя тьюрмит автоматом, графическое изображение которого находится справа. Каждый кружок представляет собой состояние, в котором находится в данный момент «мозг» тьюрмита, а стрелочки между состояниями соответствуют действиям тьюрмита в той или иной ситуации.

Всего ситуаций, по которым «мозг» тьюрмита принимает решения, две – есть ли яблоко перед ним или нет. В каждой из ситуаций тьюрмит может совершить три действия: пойти прямо и перейти в следующую клетку, повернуться вправо или влево. Если в клетке, в которую попадает тьюрмит, находится яблоко, то он съедает его.

Ваша задача состоит в том, чтобы сконструировать такой «мозг» тьюрмита, который позволит ему за 130 шагов съесть как можно больше яблок. При одинаковом числе съеденных яблок лучшей считается та конструкция автомата, которая имеет меньшее число состояний. Обратите внимание на то, что тьюрмит может сделать много шагов, находясь в одном состоянии. В этом случае автомат будет иметь петлю, начинающуюся и заканчивающуюся в этом состоянии.

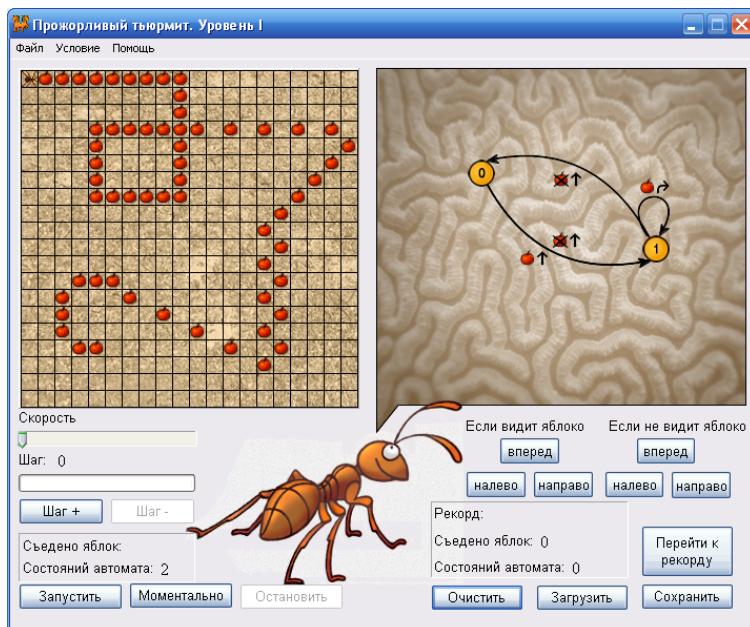


Рис. 1.2. Прожорливый тьюрмит

**Освещение города.** Демонстрация полностью настраиваемых окружений типа «теорема». Муниципалитет маленького городка хочет сэкономить на освещении улиц. Помогите ему разработать схему освещения, расставив фонари так, чтобы они осветили все закоулки города, но число фонарей при этом было минимальным (рис. 1.3).

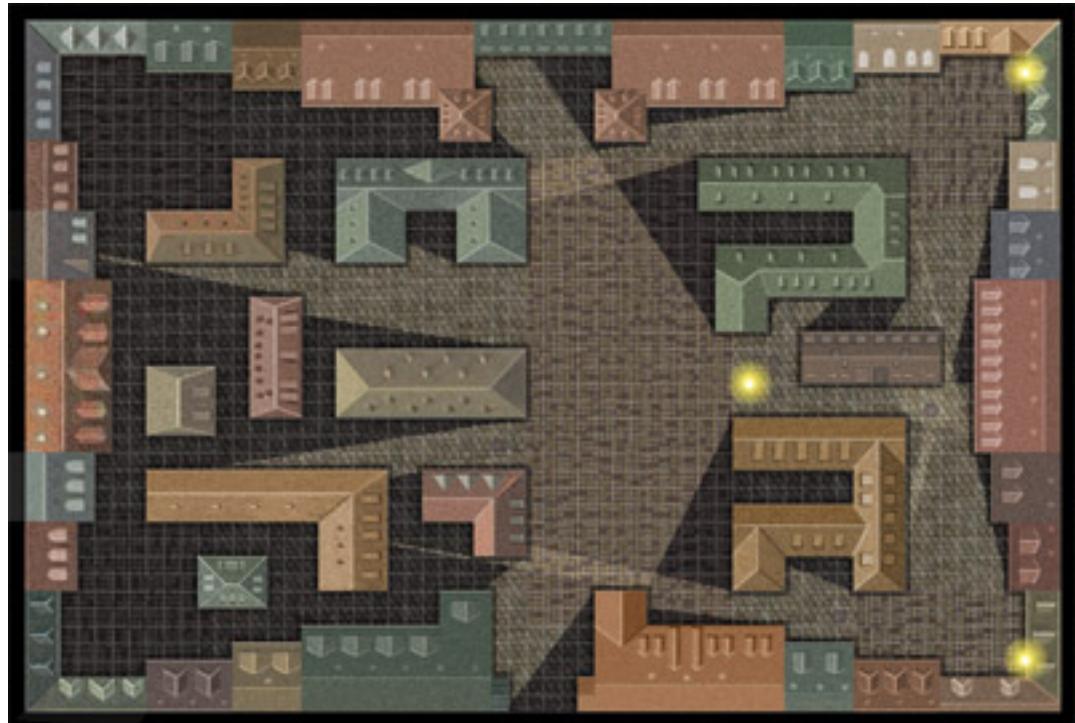


Рис. 1.3. Освещение города

**Сезам, открайся!** От каждого путника, желающего добыть новое сокровище горы Сезам, Дух Горы требует принести несколько серебряных монет. По своему усмотрению Дух обращает часть монет в золотые монеты желтого золота, а остальные - в монеты красного золота, которые немного тяжелее монет из желтого золота. При этом Дух никогда не превращает все монеты в золотые одного вида. Путника, принесшего монеты, Дух испытывает задачей. Он просит не более четырёх раз положить равное число монет на разные чашки весов так, чтобы после превращения монет в золотые равно-

весие весов хотя бы раз нарушилось. При этом хитрый Дух всегда старается так превращать монеты в два вида золотых, чтобы при всех взвешиваниях весы оставались в равновесии. Вы должны перехитрить Духа и указать такой алгоритм взвешиваний, при котором для выбранного вами числа монет равновесие всегда нарушается (рис. 1.4).

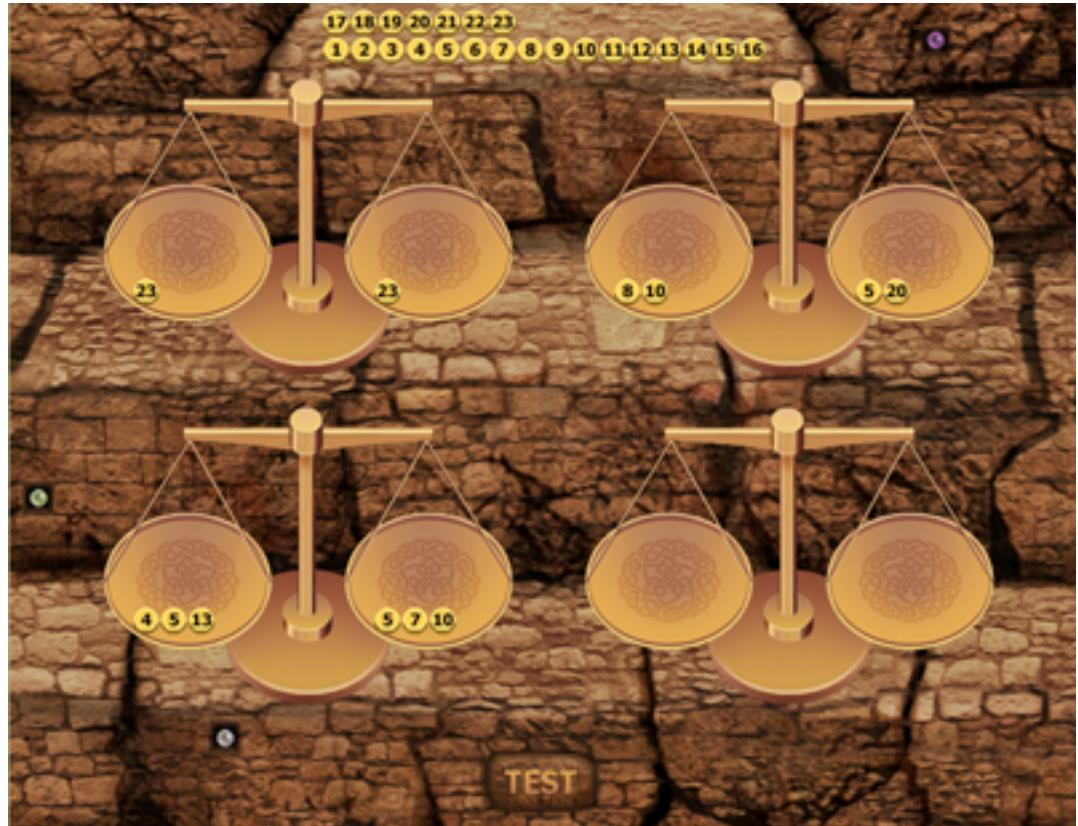


Рис. 1.4. Сезам, открайся

**Математическое скалолазание.** Известно, что многие известные ученые занимались альпинизмом. Однажды они устроили соревнование по скалолазанию по необычным правилам. Одна команда забивает в скалу 16 крючьев, а другая соединяет крючья веревкой без пересечений так, чтобы потом как можно скорее пройти по ней весь маршрут. Считается, что по всем участкам маршрута скалолазы движутся с одной скоростью, поэтому вторая команда всегда прокладывает веревку по крючьям так, чтобы минимизировать длину

маршрута. Наоборот, команда, забивающая крючья, должна позаботиться о том, чтобы даже самый короткий маршрут, проложенный по ним, был как можно длиннее. Ваша задача – вбить крючья так, чтобы максимально усложнить задачу сопернику(рис. 1.5).

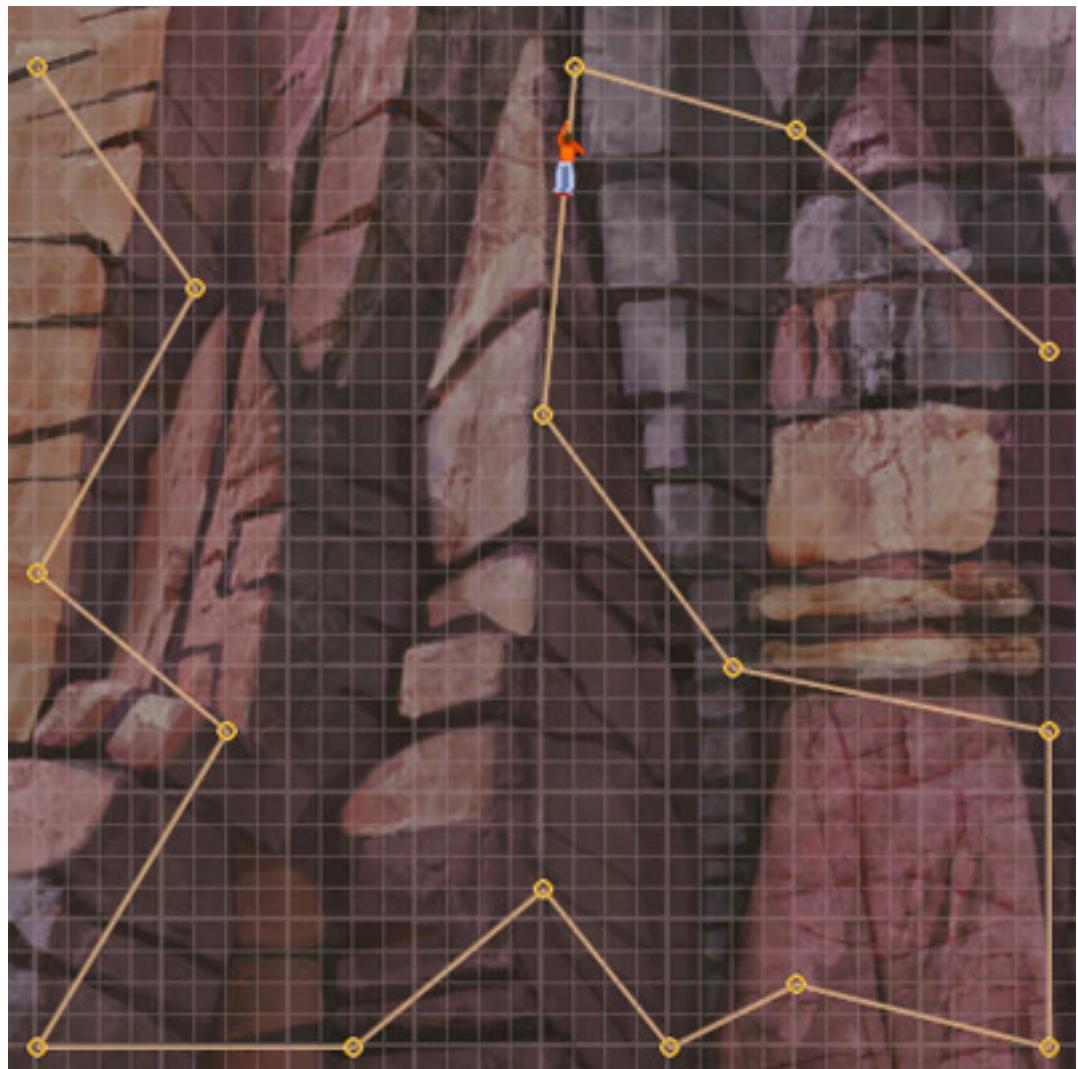


Рис. 1.5. Математическое скалолазание

До 2011 года все задачи создавались на языке Delphi. Начиная с 2011 года задачи стали делать на Flash. Flash — удобная кроссплатформенная среда для создания проектов. Так как во многих школах России теперь устанавливается СПО — свободное программное обеспечение, и, соответственно, ставится Unix подобные системы, то конкурс в виде exe-файла перестал быть легкодоступ-

пен для всех желающих участников. Для Flash-приложений же достаточно лишь наличие браузера и Flash Player'a. В этом году появилась возможность использовать динамическую библиотеку, созданную в ИТМО, которая позволяет облегчить серьёзные математические вычисления, которые приходится производить при создании подобных проектов. Возникла проблема в том, что Flash не может напрямую использовать динамические библиотеки.

## 1.2. Постановка задачи дипломного проекта

При создании различных приложений часто требуются математические и другие пакеты, которые уже реализованы в виде динамических библиотек, и создавать их заново, используя, например, Flash, слишком затратно. То есть существует потребность использовать из Flash-приложений уже имеющиеся динамические библиотеки. Решение данной проблемы касается не только Flash-приложений, а любых других платформ, используемых для создания клиентской части веб-приложения (Javascript, SilverLight, PHP, Python, Ruby). Напрямую из Flash нет возможности обращаться к динамическим библиотекам. Цель диплома - разработать компоненты, которые позволили бы использовать динамический библиотеки из Flash. Если был бы доступен исходных код библиотек, то можно было бы его адаптировать к вызову из Flash. В данном случае это будет неуниверсальный способ и при любых изменениях в динамической библиотеке, адаптацию придётся проводить заново. Нужно найти универсальный способ, который будет работать при подключении любой динамической библиотеки. Требованию к разрабатываемому продукту:

- Время на вызов функции через созданные в дипломе модули не должно отразиться на общей скорости приложения. Замедление работы программы не должно быть заметно пользователю;

- Весь проект должен занимать как можно меньше места, так как его будут скачивать участники со всех уголков России, у которых пропускная способность интернет-канала может быть слишком слабой для больших файлов. То есть увеличение размера не должно превышать нескольких мегабайт.
- Проект должен работать как под Windows, так и в различных дистрибутивах Linux;
- Из динамической библиотеки требуется вызывать функции. Каждая функция задаётся ее именем, типом возвращаемого значения, и типами аргументов.
- Исходный код должен быть понятен и прокомментирован.

## Глава 2

# Используемые технологии

## 2.1. Структура проекта

Так как исходных кодов библиотек нет, значит возможности перевести их на ActionScript нет. Было принято решение использовать виртуальный сервер для связи Flash с динамическими библиотеками. На рис. 2.1 представлена схема работы нашего проекта.

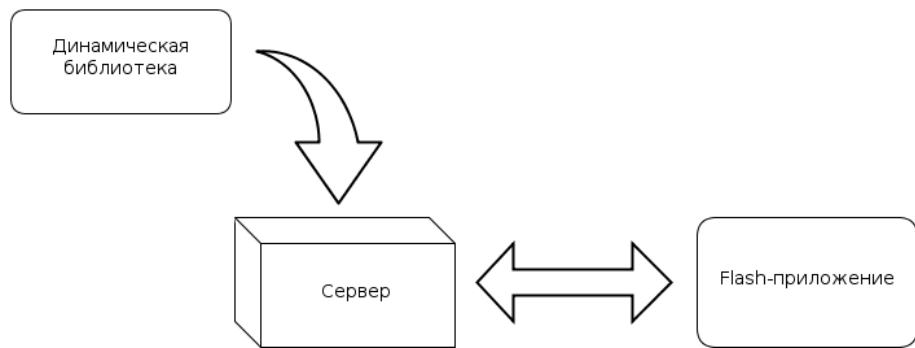


Рис. 2.1. Схема связи Динамической библиотеки и Flash-приложения

Этапы запуска приложения:

- Запуск сервера,
- Инициализация динамической библиотеки,
- Запуск Flash-приложения;

После этого Flash-приложение посылает запрос серверу на выполнение той или иной, загруженной из динамической библиотеки, функции. Сервер обрабатывает запрос, выполняет нужную функцию с полученными из запроса

параметрами, и посыпает ответ клиенту (Flash-приложению). Данная схема позволяет Flash-приложениям использовать динамические библиотеки.

## 2.2. Выбор платформы для Flash

В качестве среды программирования мною был использована среда IntelliJ IDEA (рис. 2.2). Она была выбрана, так как является удобной платформой



Рис. 2.2. Эмблема IntelliJ IDEA

для создания приложений, в ней существует большой набор мощных инструментов, не навязывающих определенных структур проекта, имеет удобный редактор. В качестве компилятора был выбран Flex SDK. По сравнению с компилятором Adobe Flash, он имеет удобный debugger, сохраняя при этом все возможности Flash. В отличие от Adobe Flash Flex SDK распространяется бесплатно.

## 2.3. Выбор платформы для создания сервера

### 2.3.1. Протокол передачи

В качестве протокола передачи информации был выбран http-протокол, потому что он, как правило, не должен перекрываться для передачи информации. Это происходит, потому что для открытия веб-страницы в интернете требуется этот протокол, следовательно ни в какой операционной системе он не запрещён к использованию. Также в ActionScript существует класс

URLLoader, который относится к пакету flash.net. Данный класс может создавать http-запросы и принимать их в виде текста, двоичных данных или переменных в кодировке URL. Методы данного класса полностью удовлетворяют потребностям проекта.

### 2.3.2. Среда создания

Сервер может располагаться локально на компьютере участника, либо находиться удаленно. Было принято решение размещать сервер локально на каждом компьютере, на котором будет устанавливаться пакет конкурса КИО. Плюсы данного варианта заключаются в том, что участникам не нужно постоянное устойчивое соединение с удалённым сервером для посылки запросов. Тем самым, ученики могут скачать установщик и участвовать в конкурсе, пользуясь компьютерами без выхода в интернет. Минус — нужно найти такой сервер, который не будет занимать много места, будет быстро скачиваться, и будет легок в настройке, быстр в использовании. Существует большое количество платформ и языков программирования для создания серверов, а именно: C++, PHP, node.js, python. Рассмотрим примеры создания серверов на этих платформах.

**Сервер на C++** В листинге 2.1 реализован простой http-сервер, который на успешный запрос клиента отвечает, что сообщение было получено. Программисту, который никогда не сталкивался с созданием серверов будет достаточно сложно разобраться в назначении многих параметров и вообще структуры создания сервера на C++.

В тоже время сервер на C++ является хорошим вариантом для нашего проекта, так как он не будет занимать много места на жёстком диске и будет достаточно быстр при обработке для наших условий.

Листинг 2.1. Пример http-сервера на C++

```
1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <string.h>
4 #include <unistd.h>
5 #include <sys/types.h>
6 #include <sys/socket.h>
7 #include <netinet/in.h>
8
9 void error(const char *msg) {
10     perror(msg);
11     exit(1);
12 }
13
14 int main(int argc, char *argv[]) {
15     int sockfd, newsockfd, portno;
16     socklen_t clilen;
17     char buffer[256];
18     struct sockaddr_in serv_addr, cli_addr;
19     int n;
20     if (argc < 2) {
21         fprintf(stderr, "ERROR, no port provided\n");
22         exit(1);
23     }
24     sockfd = socket(AF_INET, SOCK_STREAM, 0);
25     if (sockfd < 0)
26         error("ERROR opening socket");
```

```

27    bzero( (char *) &serv_addr , sizeof(serv_addr)) ;
28    portno = atoi( argv[1]) ;
29    serv_addr.sin_family = AF_INET;
30    serv_addr.sin_addr.s_addr = INADDR_ANY;
31    serv_addr.sin_port = htons( portno) ;
32    if ( bind( sockfd , (struct sockaddr *) &serv_addr ,
33                  sizeof(serv_addr)) < 0)
34        error( "ERROR_on_binding" );
35    listen( sockfd , 5) ;
36    clilen = sizeof(cli_addr);
37    newsockfd = accept( sockfd ,
38                      (struct sockaddr *) &cli_addr ,
39                      &clilen);
40    if ( newsockfd < 0)
41        error( "ERROR_on_accept" );
42    bzero( buffer , 256) ;
43    n = read( newsockfd , buffer , 255) ;
44    if ( n < 0) error( "ERROR_reading_from_socket" );
45    printf( "Here_is_the_message: %s\n" , buffer ) ;
46    n = write( newsockfd , "I_got_your_message" , 18) ;
47    if ( n < 0) error( "ERROR_writing_to_socket" );
48    close( newsockfd ) ;
49    close( sockfd ) ;
50    return 0;
51 }
```

**Сервер на PHP** Теперь рассмотрим пример http-сервера на PHP. Последняя версия PHP 5.4 содержит встроенный веб-сервер, соответственно не требуется установка Apache или других аналогичных серверов, которые сложны в настройке. Запускается сервер следующим образом. Пример для Linux:

```
$ php -S localhost:8000
```

В консоли выводится следующее:

```
PHP 5.4.0 Development Server started at Date
Listening on localhost:8000
Document root is /home/username/
Press Ctrl-C to quit
```

Скрипт index.php будет выглядеть так:

```
<?
echo "Hello world!" ;
```

На каждый запрос, посланный серверу, будет выдаваться содержимое index.php, в данном примере будет выводиться Hello world!

**Сервер на Python** В коде листинга 2.2 сервер обрабатывает запрос, и при успешном запросе выдаёт клиенту запрошенную страницу.

Листинг 2.2. Пример http-сервера на Python

```
1 import socket
2 import sys
3
4 def send_string(socket, data):
5     bytes_to_send = len(data)
6     while bytes_to_send > 0:
7         sent_bytes = socket.send(data, bytes_to_send)
```

```

8     if sent_bytes == -1:
9         return False;
10    bytes_to_send == sent_bytes
11    return True
12
13 def recv_line(socket):
14     EOL = "\r\n"
15     nbytes = 1
16     buffer = socket.recv(nbytes)
17
18     while not EOL in buffer:
19         buffer += socket.recv(nbytes)
20     result = buffer[:-2]
21     return (result, len(result))
22
23 ROOTDIR = "./www"
24
25 def process_connection(sock, address):
26
27     request, length = recv_line(sock)
28     print 'Got request from %s:%d "%s"\n'
29             % (address[0], address[1], request)
30
31     if "HTTP/" in request:
32         file_path = None
33         if "GET_" in request:
34             file_path = request[4:-9]

```

```

35     if "HEAD_" in request:
36         file_path = request[5:-9]
37
38     if file_path:
39         if file_path == '/':
40             file_path = '/index.html'
41         resource = ROOTDIR
42         resource += file_path
43     try:
44         f = open(resource, 'rb')
45         print "200 OK\n"
46         send_string(sock, "HTTP/1.0 200 OK\r\n")
47         send_string(sock, "Content-Type: text/html; charset"
48                     ="UTF-8\r\n")
49         send_string(sock, "Set-Cookie: name=value\r\n")
50                         # install cookies
51         send_string(sock, "Server: NanoPyHttpd\r\n\r\n")
52     if "GET_" in request:
53         file_content = f.read()
54         sock.send(file_content, 10485760)
55                         # problem with sending huge files
56
57         f.close()
58     except IOError:
59         print "404 Not found\n"
60         send_string(sock, "HTTP/1.0 404 NOT FOUND\r\n")
61         send_string(sock, "Server: NanoPyHttpd\r\n\r\n")

```

```

61         send_string(sock , "404_Not_Found")
62         send_string(sock , "URL_not_found\r\n")
63
64     else :
65         print "\tUnknown_request!\n" # if type of request is
66             unknown
67
68     else :
69         print "NOT_HTTP!\n"
70
71     sock.shutdown(socket.SHUT_RDWR) # closing the socket
72     sock.close()
73
74
75 def main():
76     host = ''
77     port = 8080
78     backlog = 5
79     s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
80     s.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
81     s.bind((host, port))
82     s.listen(backlog)
83     while 1:
84         client, address = s.accept()
85         process_connection(client, address)
86
87
88 if __name__ == '__main__':
89     main()

```

Сервер на Python умеет отвечать на запрос с именем страницы выводом самой страницы.

**Сервер на Node.js** Node.js — это сравнительно новая платформа, построенная на основе Chrome's JavaScript Runtime для легкого создания быстрых, масштабируемых сетевых приложений. Node.js позволяет выполнять Javascript-код вне браузера. Он состоит из среды исполнения программ и большого количества дополнительных библиотек. Листинг 2.3 отражает код для создания сервера на Node.js

Листинг 2.3. Пример http-сервера на Node.js

```
1 var http = require("http");
2 http.createServer(function(request, response) {
3     response.writeHead(200, {"Content-Type": "text/plain"});
4     response.write("Hello World");
5     response.end();
6 }).listen(8888);
```

Все. Вот и весь сервер. На каждый запрос он выводит Hello World. Очень просто и понятно. К неоспоримым плюсам относится то, что процесс node.js живёт долго и обрабатывает все http-запросы внутри себя. Это значит, что для каждого нового запроса не выполняется инициализация, как, например, на PHP. А это существенно сокращает время обработки запросов. С помощью таких инструментов как The Node Toolbox (<http://toolbox.no.de/>) и npm (<http://npmjs.org/>) процесс поиска, выбора и установки необходимых модулей становится простым и быстрым. Также очень понравилась документация, которая существует не только на английском, но и на русском языках.

Заглядывая в будущее, если предположить, что сервер будет запускаться удалённо, тогда не нужна никакая настройка сервера локально на каждом компьютере. Но тогда будет возникать проблема перегруженности сервера, ведь в предыдущие годы в конкурсах участвовали более 60 тысяч учеников и их число каждый год растёт.

Node.js выполняет все запросы асинхронно. Если какой-то запрос приходит на сервер, а сервер уже занят обработкой другого процесса, то новый запрос всё равно сразу же начнёт выполняться. Быстродействие node.js впечатляет.

Приведу сравнительный анализ, представленный на сайте <http://b.brainscode.com/2011/04/nodejs-vs-php.html>. В этом анализе сравнивается быстродействия node.js и php. Проведем тесты на простых математических вычислениях. В листингах 2.4 и 2.5 представлены скрипты тестов на PHP и Node.js соответственно.

Листинг 2.4. Тестовый скрипт php

```
1 $a = null;
2 $b = null;
3 $c = null;
4 $i = null;
5 $max = 1e6;
6 $start = microtime(true);
7 for ($i = 0; $i < $max; $i++) {
8     $a = 1234 + 5678 + $i;
9     $b = 1234 * 5678 + $i;
10    $c = 1234 / 2 + $i;
11 }
12 var_dump(microtime(true) - $start);
```

Листинг 2.5. Тестовый скрипт Node.js

```
1 var i, a, b, c, max;
2 max = 1e6;
3 console.time('maths');
```

```

4 | for ( i = 0; i < max; i++) {
5 |     a = 1234 + 5678 + i ;
6 |     b = 1234 * 5678 + i ;
7 |     c = 1234 / 2 + i ;
8 |
9 | console . timeEnd ( 'maths' );

```

Таблица 2.3.2 показывает результаты тестов.

	Время (сек.)	
Кол-во итераций	Php	Node.js
100 000	0.077	0.02
1 000 000	0.759	0.016
10 000 000	7.605	0.157
100 000 000	75.159	1.567

Таблица 2.3.2. Сравнение быстродействия PHP и Node.js

Как видим nodejs существенно превосходит в данном teste, при самой большой размерности выигрыш в времени выходит почти в 50 раз, при значении в 100 000 раз этот показатель равен — 38 соответственно. Выигрыш, как видно существенный. Также в видео-лекции Ryan'a Dahl'a (одного из создателей Node.js) показано, что если создан сервер, который на каждый запрос отвечает двух-секундной задержкой (см. 2.6), то при 100 000 одновременных запросах время обработки запросов сервером будет составлять чуть более 2 секунд, что свидетельствует о параллельной обработке запросов, об асинхронной работе сервера.

Листинг 2.6. Тело функции обработки запроса в Node.js

```

setTimeOut ( function () {
    console . log ( " hello _ world " );
}

```

| } , 2000);

И, в конце-концов, программы на Javascript являются легко читаемыми.

По вышеперечисленным причинам было принято решение использовать сервер, созданный на node.js. Из минусов можно отметить, что сервер будет занимать чуть менее 5 мб, но, учитывая все достоинства использования Nodejs, этим минусом можно пренебречь. Даже если предположить, что у участника интернет типа dial-up с максимальной скоростью 64 кбит/сек:

- 64 кбит/сек = 8 кбайт/сек;
- 5 мбайт = 5120 кбайт (максимальный размер сервера)
- $5120 / 8 = 640$  сек;
- $640 / 60 = 10$  минут 40 секунд - худший расклад.

При скорости 512 кбит/сек (64 кбайт/сек) время закачки будет равно 80 секундам.

### 2.3.3. Создание тестовой динамической библиотеки

Также для создания тестовой динамической библиотеки будет использоваться Microsoft Visual Studio 2010 Express для Windows и компилятор gcc на Linux. В листинге 2.7 приведен код, из которого компилируется динамическая библиотека.

Листинг 2.7. Код компилируемой динамической библиотеки

```
1 int add(int a, int b)
2 {
3     return a + b;
4 }
5
```

```
6 int multiplication(int a, int b){  
7     return a * b;  
8 }  
9 int subtraction(int a, int b){  
10    return a - b;  
11 }
```

Чтобы создать динамическую библиотеку в Linux нужно сначала скомпилировать файл big.c. Для этого в терминале вводим следующее:

```
gcc -fPIC c← big.c
```

Ключ -fPIC включен, потому что мы создаём динамическую библиотеку. Это нужно для того, чтобы переходы в функциях использовали не абсолютную адресацию, а относительную, иначе несколько различных программ не смогут использовать эту библиотеку. В результате мы получаем файл big.o, который используем следующим образом:

```
gcc -shared -o big.o
```

В результате будет получен файл big.so, который и будет являться динамической библиотекой, используемая далее в проекте.

## 2.4. Структура проекта

После того как все языки программирования были выбраны, структура проекта будет иметь вид как на рис. 2.3. К Node.js подключается дополнительная библиотека node-ffi для работы с динамическими библиотеками. Общение Flash-приложения с сервером происходит через http-запросы и http-ответы. Библиотека node-ffi является дополнительной библиотекой Node.js, которая работает с динамическими библиотеками. Принцип ее работы описан в разделе 4.3.1.

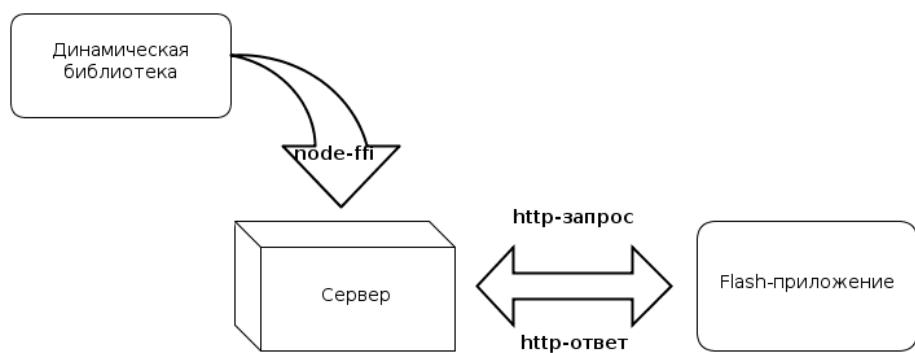


Рис. 2.3. Структура проекта со связями

# Глава 3

## Проектирование

### 3.1. Настройка Node.js

#### 3.1.1. Настройка под Linux

В данном разделе описывается установка Node.js на Ubuntu 11.10. Устанавливаем сам nodejs:

```
$ sudo apt-get install nodejs
```

Далее устанавливаем пакетный менеджер npm (Node Package Manager):

```
$ sudo apt-get install npm
```

Перед установкой node-ffi должен быть установлен Python (он уже по умолчанию стоит на Ubuntu 11.10). Если требуется установка, то также используется пакетный менеджер (рекомендуемая версия Python 2.7.2). Также должен быть установлен C/C++ компилятор, например, gcc:

```
$ sudo apt-get install gcc
```

После предварительных установок устанавливаем node-ffi:

```
$ npm install ffi
```

Если установка не прошла успешна, а все предыдущие шаги были выполнены верно, то значит в операционной системе используется библиотека libffi не той версии, которая нам нужна. Такая проблема возникла в Ubuntu 11.10. При установке на Ubuntu 12.04 такой проблемы нет. Соответственно тогда удаляем старую библиотеку:

```
$ sudo apt-get remove libffi4
```

И устанавливаем (libffi-dev, если libffi нет):

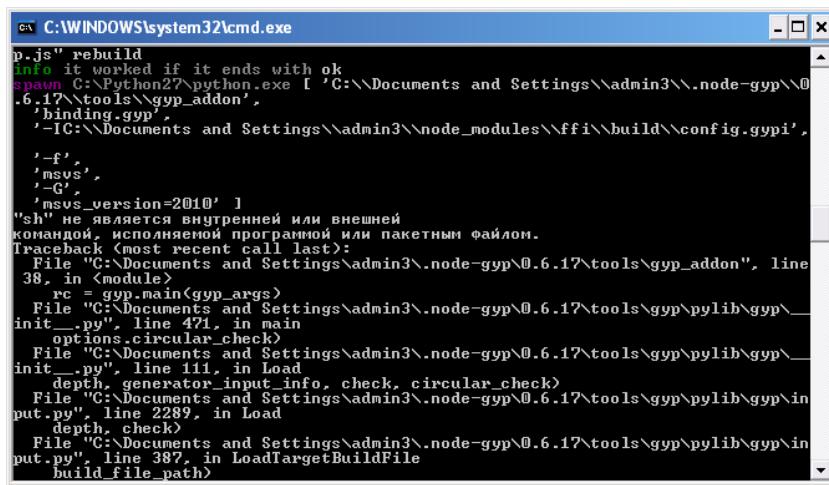
```
$ sudo apt-get install libffi
```

После этого снова повторяем установку node-ffi.

### 3.1.2. Настройка под Windows

Для установки Node.js достаточно скачать установщик с официального сайта программы ([nodejs.org](http://nodejs.org)) и установить его. На данном этапе проекта подключить node-ffi к проекту в Windows не удалось. Далее представлены испробованные способы установки.

**Установка, используя документацию, приложенную к библиотеке node-ffi.** Предварительно под Windows нужно установить python 2.7.2, также Microsoft Visual Studio C++, разработчики рекомендуют последнюю версию на данный момент, а именно Visual C++ 2010 Express. Также для 64-х битных систем нужно установить Windows 7 64-bit SDK. После установки всего в командной строке была произведена попытка установки node-ffi через npm (npm install node-ffi). На рис. 3.1 представлена ошибка, возрастающая при таком способе установки. SH — Shell Script — скрипт, в котором представле-



The screenshot shows a Windows Command Prompt window titled 'cmd C:\WINDOWS\system32\cmd.exe'. The window contains the following text:

```
p.js" rebuild
info it worked if it ends with ok
spawn C:\Python27\python.exe [ 'C:\Documents and Settings\admin3\.node-gyp\0.6.17\tools\gyp_addon',
  '-binding.gyp',
  '-IC:C:\Documents and Settings\admin3\node_modules\ffi\build\config.gypi',
  '-f',
  'msvs',
  '-G',
  'msvs_version=2010' ]
"sh" не является внутренней или внешней
командой, исполняемой программой или пакетным файлом.
Traceback (most recent call last):
  File "C:\Documents and Settings\admin3\.node-gyp\0.6.17\tools\gyp_addon", line
  38, in <module>
    rc = gyp.main(gyp_args)
  File "C:\Documents and Settings\admin3\.node-gyp\0.6.17\tools\gyp\gyp\__init__.py", line 471, in main
    options.circular_check>
  File "C:\Documents and Settings\admin3\.node-gyp\0.6.17\tools\gyp\gyp\__init__.py", line 111, in Load
    depth, generator_input_info, check, circular_check>
  File "C:\Documents and Settings\admin3\.node-gyp\0.6.17\tools\gyp\pylib\gyp\__init__.py", line 2289, in Load
    depth, check>
  File "C:\Documents and Settings\admin3\.node-gyp\0.6.17\tools\gyp\pylib\gyp\__init__.py", line 387, in LoadTargetBuildFile
    build_file_path>
```

Рис. 3.1. Ошибка при установке node-ffi через npm в командной строке

ны наборы команд для последовательного выполнения команд в командной

строк Unix-подобных систем. Соответственно, под Windows данная команда на выполнение скрипта не будет выполняться.

**Установка с помощью "Установщика веб-платформы".** Данная программа является аналогом пакетного менеджера в Unix-подобных системах. Скачать её можно на сайте:

<http://www.microsoft.com/web/downloads/platform.aspx> (рис. 3.2) Для уста-

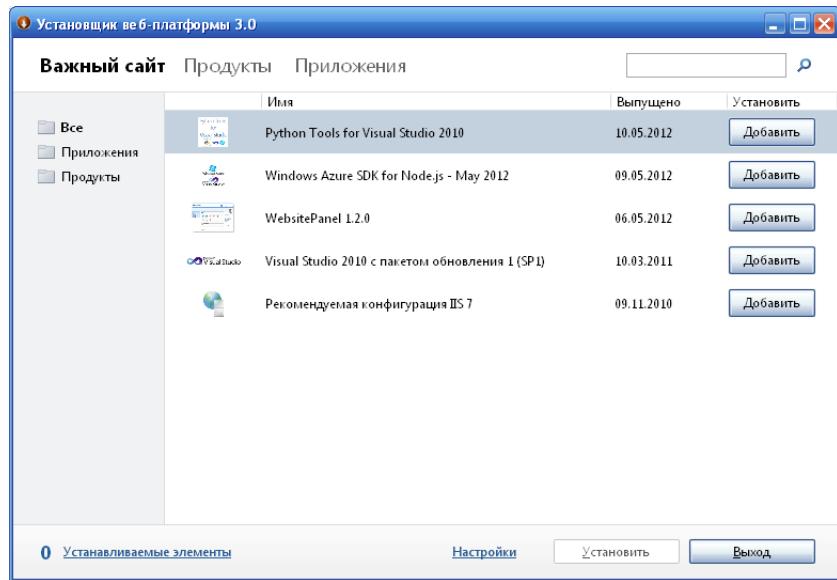


Рис. 3.2. Установщик веб-платформы 3.0

новки node.js через установщик нужно установить рабочее окружение. Устанавливаем рекомендуемую конфигурацию IIS 7. Заходим в Настройки и добавляем сайт, на котором можно скачать nodejs:

<http://www.helicontech.com/zoo/feed> (рис. 3.3). После этого на вкладке Zoo устанавливаем в разделе Packages, как показано на рис. 3.4. Далее устанавливаем WebMatrix Templates. На рис. 3.5 отображена возвращающаяся ошибка при установке. Никаких советов по исправлению этой проблемы найти не удалось. Установка таким способом пробовалась на Windows XP sp2 x64, sp3 x64 и x86. При удалении текущих версий .NET Framework никаких положительных изменений не происходило. Поэтому данный метод также оказался неудачным.

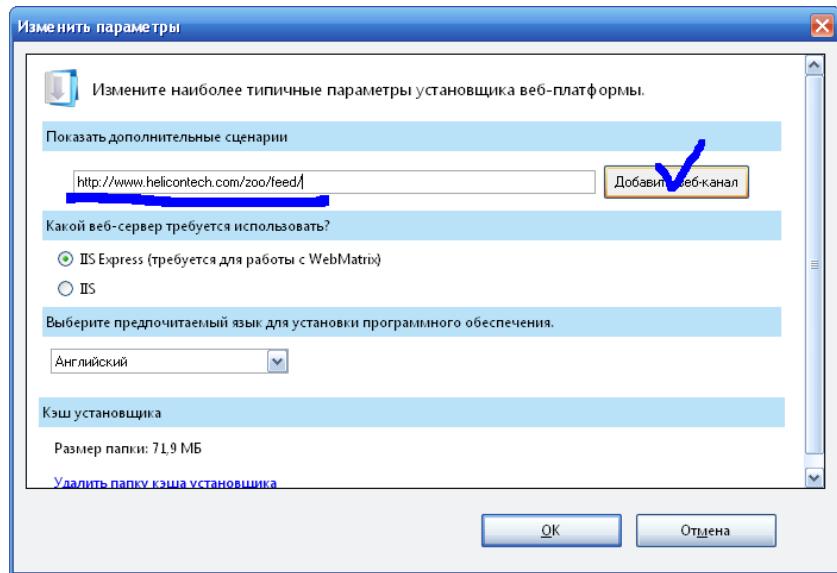


Рис. 3.3. Добавление сайта, содержащего пакет установки Node.js

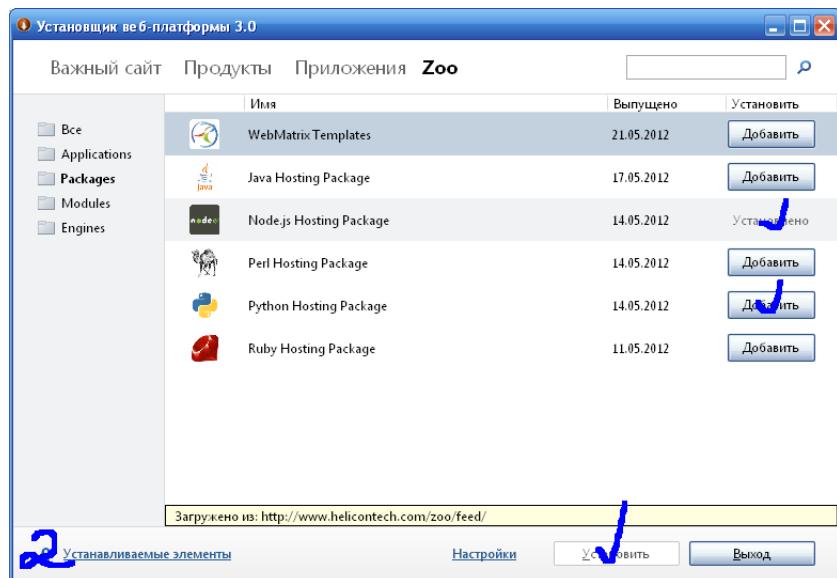


Рис. 3.4. Установка требуемых пакетов

**Установка Node.js через Cygwin.** Далее появилась идея избежать ошибки, выдаваемой командной строкой о выполнении sh, установкой node-ffi через Cygwin. Cygwin — UNIX-подобная среда и интерфейс командной строки для Microsoft Windows. Cygwin обеспечивает тесную интеграцию Windows приложений, данных и ресурсов с приложениями, данными и ресурсами UNIX-подобной среды. Из среды Cygwin можно запускать Windows приложения,

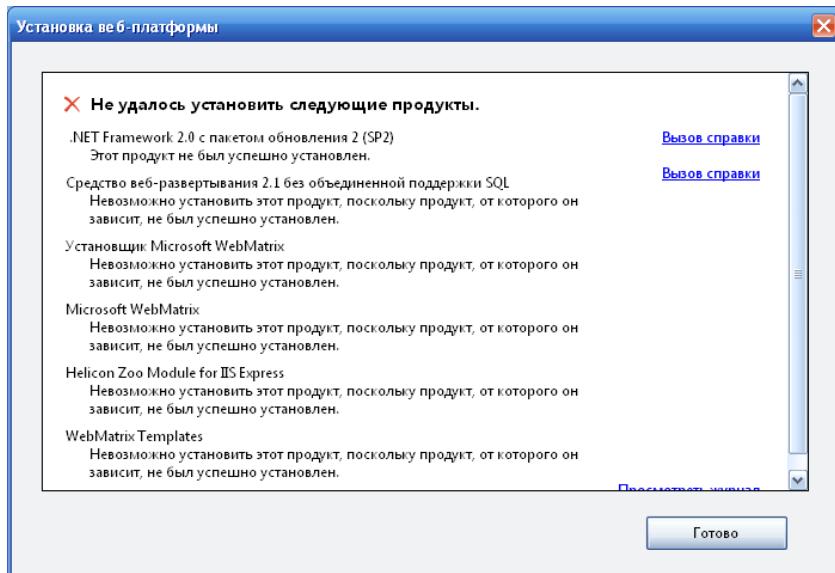


Рис. 3.5. Возвращаемая ошибка

также можно использовать инструменты Cygwin из Windows.

Cygwin состоит из двух частей: динамически подключаемая библиотека cygwin1.dll, которая обеспечивает совместимость API и реализует значительную часть стандарта POSIX и огромная коллекция приложений, которые обеспечивают привычную среду UNIX.

Стандарт Posix – это один из общепринятых стандартов повышения мобильности программного обеспечения (ПО). Задача обеспечения мобильности ПО является очень важной задачей исключительной сложности. Едва ли стоит обосновывать это каким-либо более широким способом. При установке Cygwin в каталог собирается Linux-подобный корневой каталог, представленный на рис. 3.6.

При запуске Cygwin-Terminal открывается Terminal как в Linux, соответственно, если в его окружение установить всё, что надо (смотри пункт 3.1.1), то должен установиться node-ffi. Но возникла проблема с тем, что установка libffi тянет за собой, грубо говоря, весь Linux. Очень много времени было потрачено на попытки установки node-ffi через Cygwin, но результат не был

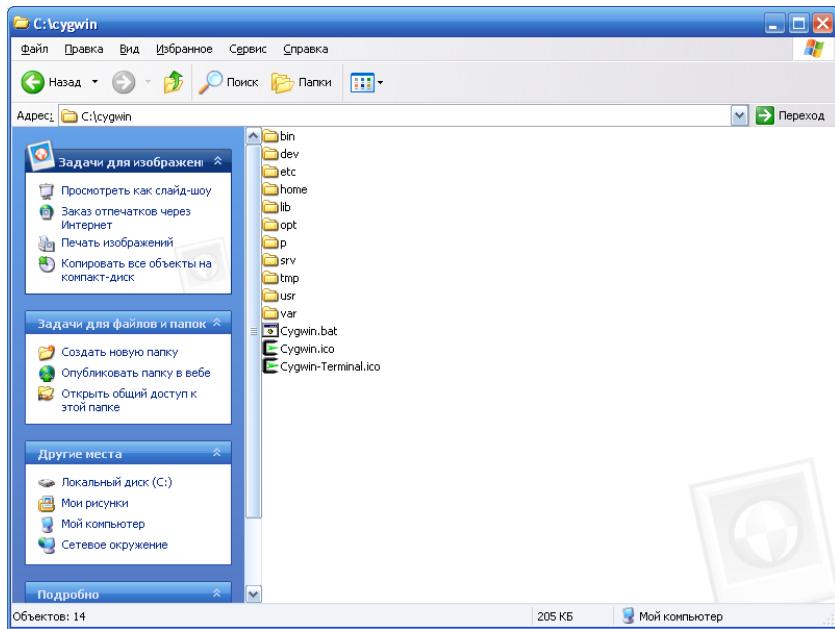


Рис. 3.6. Корневая папка Cygwin

достигнут.

**Подведение итогов.** На данный момент решение данной проблемы придумано не было. Создателем платформы был отправлен запрос на помощь в установке node-ffi с подробным описанием проблем в установке.

Буквально на днях был получен ответ, в котором было сказано попробовать установить всё через Mozilla-Build, при этом скомпилировав в ней libffi. В условиях сдачи диплома на данный момент времени попробовать такой вариант нет. Работа над установкой node-ffi под Windows будет продолжена позднее. Вообще говоря, большое количество программистов, использующих node.js сетуют на Node Package Manager, потому что он требует серьёзных предустановок и даже при этом не работает стабильно.

Существует предложение создать уже скомпилированные файлы для установки под каждую версию операционных систем, для того, чтобы не нужно было устанавливать такой большой объем программ. Кстати, в рамках нашего проекта, это может послужить большой проблемой, размещая сервер

локально на каждом компьютере, будет необходимо установить большой объём информации. С одной стороны технология использования Node.js пока что не оправдала себя полностью , но ещё есть более полугода, возможно, разработчики исправят эту проблему, либо придётся компилировать всё своими руками. С другой стороны, размещение сервера удалённо решит эту проблему и тут node.js, как уже было описано, подходит как нельзя кстати.

Нет волнений, что текущая идея может быть невыполнима, используя выбранные технологии.

## Глава 4

# Реализация программы

## 4.1. Предварительная работа с динамической библиотекой

Для того, чтобы при изменении динамической библиотеки не нужно было менять код программы было сделано следующее. В корневом каталоге с файлами, необходимыми для создания сервера, создаётся текстовый файл base.txt. Содержание файла формируется из информации о функциях, заявленных в динамической библиотеке. Каждая функция записывается в текстовый файл следующим образом:

```
"function_name","return_type","var's_type",["var's_type",
[...]] ,
```

Сначала записывается имя функции, потом тип возвращаемого значения, далее перечисляются типы передаваемых переменных в функцию в том порядке, в котором они идёт в ней. В конце записи ставится запятая. Следующая функция записывается со следующей строки. После того как все функции описаны, на следующей строке записывается имя динамической библиотеки без расширения. После имени никаких символов не ставится. К созданной динамической библиотеке (см. пункт 2.3.3) файл base.txt будет выглядеть следующим образом:

```
1 add,int,int,int,
2 multiplication,int,int,int,
3 subtraction,int,int,int,
4 big
```

В пункте 4.3 описано, как используется этот файл.

## 4.2. Оформление запроса

Flash-приложение посылает запрос на сервер. Сервер разбирает этот запрос, выполняет нужную функцию из динамической библиотеке, и посыпает результат выполнения ответов. Разберем, как организовать запрос на примере:

```
"http://127.0.0.1:8888/>?int=subtraction+float=123.3+int=1"
```

- `http://127.0.0.1:8888/` - сайт, куда посыпается запрос с портом, который прослушивает сервер;
- `>?` - данный символ указывает серверу на то, что этот запрос - это запрос на выполнение функции из подгружаемой динамической библиотеки;
- `int=subtraction` - тип возвращаемого значения и имя функции;
- `float=123.3` - тип передаваемого параметра и его значения;

Между параметрами и описанием функции ставится `+`. Все передаваемые параметры должны быть перечислены строго в том порядке, в котором они заданы в функции из динамической библиотеки. Тип параметров должны в точности совпадать с типами передаваемых переменных, объявленных в функции. На данном этапе разработки при неправильно созданном запросе, сервер пытается найти функцию с такими параметрами и, не находя её, аварийно завершает работу. Существует следующие варианты того, как будет вести себя сервер при неверном запросе:

- Пропускать данный запрос, не уведомляя об ошибке запроса.
- Возвращать ошибку.

Для программиста, который пишет Flash-программу, если он неверно задаёт запрос, для отладки важно, чтобы ему возвращалась ошибка, с указанием о том, какой запрос был создан неправильно. Поэтому выбран второй вариант. При этом важно сделать так, чтобы Flash-программисту было легко понять какой из запросов выдал ошибку, а также охарактеризовать тип ошибки, предложив пути решения. Данная проблема будет решена в следующих версиях моего проекта.

### 4.3. Структура проекта

Весь проект делится на 2 части:

- Написать сервер, работающий с динамической библиотекой.
- Написать функцию в Flash для работы с запросом

#### 4.3.1. Серверная часть

Для серверной части написано три скрипта:

- http.js - Создание сервера;
- readRequest.js - Разбор полученного запроса;
- text.js - Инициализация динамической библиотеки.

В text.js написана функция, которая на основе файла base.txt (см пункт 4.1) подключает динамическую библиотеку. В readRequest.js написаны две функции. Функция GetReturnType(text) - возвращает тип возвращаемого значения функции. Функция Search(text, way) - возвращает массив, при этом если

в параметр way указать значение "type то массив будет состоять из типов передаваемых параметров. Если же передать "value массив значений, которые нужно передать в качестве параметров функции. Рассмотрим работу библиотеки node-ffi на примере нашего проекта (см. 4.1).

Листинг 4.1. Код text.js. Инициализация динамической библиотеки

```
1 var ffi = require("node-ffi");
2 var fs = require("fs");
3 function DynLib() {
4     var request = [];
5     fs.readFile('base.txt', 'UTF-8', function (err, text) {
6         if (err) throw err;
7         curPos = 0;
8         endPos = 0;
9         while (text.indexOf("\n", curPos) != -1) {
10             endPos = text.indexOf("\n", curPos);
11             if (endPos == -1)
12                 { endPos = text.length }
13             request.push(text.slice(curPos, endPos));
14             curPos = endPos + 1;
15         }
16         // console.log("req: ", request);
17         StructForLib = new Object();
18
19         for (i=0; i < request.length -1; i++) {
20             curPos = 0;
21             endPos = 0;
22             returnType = "";
```

```

23     funcName = "";
24     funcType = [];
25     endPos = request[i].indexOf(", ", curPos);
26     funcName = request[i].slice(curPos, endPos) + "";
27     curPos = endPos + 1;
28     endPos = request[i].indexOf(", ", curPos);
29     returnType = request[i].slice(curPos, endPos);
30     curPos = endPos + 1;
31     while (request[i].indexOf(", ", curPos) != -1) {
32         endPos = request[i].indexOf(", ", curPos);
33         funcType.push(request[i].slice(curPos, endPos));
34         curPos = endPos + 1;
35     }
36     StructForLib[funcName] = [returnType, funcType];
37 }
38
39 libName = "./" + request[request.length - 1];
40 lib1 = ffi.Library(libName, StructForLib);
41 console.log("StructForLib="); console.log(StructForLib);
42 });
43
44 }
45 exports.DynLib = DynLib; //externs the func to other files

```

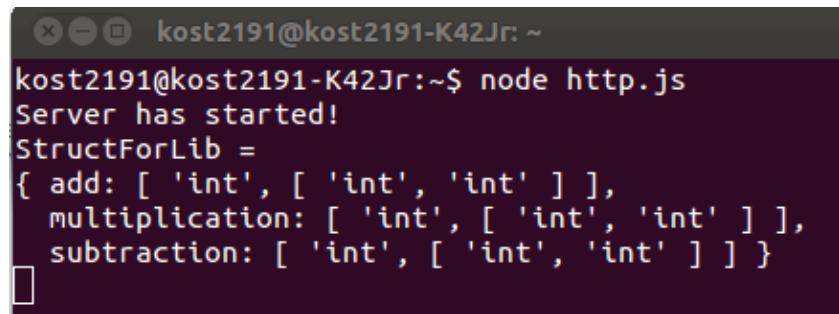
Следующие строки подключают библиотеки для работы с динамическими библиотеками и для работы с файлами соответственно:

```

1 var ffi = require("node-ffi");
2 var fs = require("fs");

```

В данном скрипте функция DynLib() формирует объект StructForLib для выгрузки из динамической библиотеки функций. Рис. 4.1 показывает, как выглядит StructForLib перед выгрузкой из тестовой динамической библиотеки: После того, как StructForLib был сформирован, по нему создается объект lib1



```
kost2191@kost2191-K42Jr:~$ node http.js
Server has started!
StructForLib =
{ add: [ 'int', [ 'int', 'int' ] ],
  multiplication: [ 'int', [ 'int', 'int' ] ],
  subtraction: [ 'int', [ 'int', 'int' ] ] }
```

Рис. 4.1. Параметр для передачи в Library из node-ffi

с методами, являющимися функциями динамической библиотеки:

```
1 | lib1 = ffi.Library(libName, StructForLib);
```

После этого в объекте lib1 станут доступны методы: add(int, int), multiplication(int, int), subtraction(int, int). Каждый раз при обработке запроса из lib1 будет вызываться метод, который был запрошен Flash-приложением.

Листинг 4.2. Код http.js. Сервер

```
1 var http = require("http");
2 var readReq = require("./readRequest");
3 var ini = require("./text");
4 var funcValue = []
5 var funcType = [];
6 var ffi = require("node-ffi");
7 var returnType = "";
8 var funcName = "";
9 var num = 1
10
```

```

11 ini.DynLib();
12 //var lib1 = ffi.Library("./lib1", {"ad": [ "int", [ "int",
13   "int" ]]} );
14 http.createServer(function(req, response) {
15   response.writeHead(200, {"Content-Type": "text/plain"});
16   req.setEncoding('utf-8');
17   if (req.url[1] == ">") {
18     funcValue = readReq.Search(req.url, "value");
19     funcType = readReq.Search(req.url, "type");
20     returnType = readReq.GetReturnType(req.url);
21     funcName = funcValue[0];
22     funcValue.shift();
23     console.log("SENDED!");
24     callMethod = "lib1." + funcName + "(";
25     for (i = 0; i < funcType.length; i++) {
26       callMethod = callMethod + funcValue[i];
27       if (i < funcType.length - 1) {
28         callMethod = callMethod + ",";
29       } else {
30         callMethod = callMethod + ")";
31       }
32     }
33     console.log(callMethod);
34     var rez = eval(callMethod) + "";
35     response.write(rez);
36     console.log("result=" , rez);

```

```
37  }
38
39  response.end();
40 }).listen(8888);
41 console.log("Server has started!");
42 b = "f1";
```

Сначала подключаются скрипты text.js, readRequest.js. Далее вызывается функция инициализации из text.js. После этого создается сервер, слушающий порт 8888. В котором каждый запрос с > обрабатывается и вызывается соответствующая функция, после чего сервер отправляет ответ, возвращаемый данной функцией.

Строки кода 23-31 переводят уже обработанный запрос в созданный на этапе инициализации метод (его вызов записывается в переменную типа String), взятый из динамической библиотеки. В него передаются аргументы из запроса. Далее функцией eval вызывается метод, записанный в строковой переменной. Таким образом может быть вызвана любая функция из динамической библиотеки.

#### 4.3.2. Клиентская часть

Клиентом у нас в данном случае является приложение, написанно на ActionScript. При настройках Flash по умолчанию, он не может отправлять и получать данные с локальных серверов. Так как наш сервер поднимается локально, значит нужно поменять настройки flash. Рассмотрим типы изолированной программной среды безопасности, в которой работает вызывающий файл.

- REMOTE: этот файл загружается с URL-адреса в Интернете и используется в соответствии с правилами изолированной программной среды

на основе домена;

- LOCAL\_WITH\_FILE: этот файл является локальным файлом, не является доверенным для пользователя и не публиковался с сетевым наименованием. Файл может считывать информацию из локальных источников данных, но не может обмениваться данными через Интернет;
- LOCAL\_WITH\_NETWORK: этот SWF-файл является локальным файлом, не является доверенным для пользователя и публиковался с сетевым наименованием. SWF-файл может обмениваться данными через Интернет, но не может считывать информацию из локальных источников данных;
- LOCAL\_TRUSTED: этот SWF-файл является локальным файлом, пользователь сделал его доверенным с помощью диспетчера настроек проигрывателя Flash Player или файла конфигурации FlashPlayerTrust. Файл может считывать информацию из локальных источников данных и обмениваться данными через Интернет;
- APPLICATION: файл работает в приложении AIR и был установлен с пакетом (файлом AIR) для этого приложения. По умолчанию файлы в изолированной программной среде безопасности приложения AIR могут выполнять перекрестные сценарии с любым файлом из любого домена (в то время как файлы за ее пределами могут не иметь разрешения на выполнение перекрестных сценариев с файлом AIR). По умолчанию файлы в изолированной программной среде безопасности приложения AIR могут загружать содержимое и данные из любого домена.

В случае работы программиста используется FlashPlayerDebugger из Flex SDK. Он позволяет эффективно отлаживать программу, в её теле ставить breakpoint'ы а также использовать функцию trace(), выводящую в консоль

IntelliJ IDEA тот параметр, который мы передали в trace(). Для смены настроек изолированной среды, нужно перейти в папку с используемым Flash Player'ом и вызвать настройки. Выбрать в настройках изолированной среды тип LOCAL\_TRUSTED и сохранить изменения.

Функция, нужная для работы программиста, представлена в листинге 4.3.

Листинг 4.3. Функция для Flash-приложения

```
1  public function UrlConnect () : void {
2      var FuncName = "multiplication";
3      var Rtype = "int";
4      var a = 1;
5      var b = 2;
6      SendRequest (FuncName , Rtype , [ a , b ] );
7  }
8
9  public function SendRequest (fname: String , Rtype:
10     String , vars: Array) {
11     var Rurl = "http://127.0.0.1:8888/>?" + Rtype + "="
12     " + fname + "+";
13     var i ;
14     for ( i =0; i < vars . length ; i ++ ) {
15         Rurl = Rurl + typeof ( vars [ i ]) + "=" + vars [ i ];
16         if ( i < vars . length - 1 ) {
17             Rurl = Rurl + "+";
18         }
19     }
20     trace ( Rurl );
```

```
19     loader.dataFormat = URLLoaderDataFormat.TEXT;
20     loader.addEventListener(Event.COMPLETE,
21         loaderHandler);
22     loader.load(new URLRequest(Rurl));
23 }
24
25 public function loaderHandler(event:Event):void{
26 }
```

Программисту для получения результата достаточно передать имя функции, тип возвращаемого значения, а также массив из аргументов. Пример этого описан в функции UrlConnect() листинга 4.3. Требования к аргументам функции SendRequest(..):

- Имя функции должно присутствовать в динамической библиотеке;
- Тип возвращаемого значение должен соответствовать типу возвращаемого значения в функции динамической библиотеки;
- Аргументы функции (массив vars) должны быть указаны в правильном порядке и соответствовать типам аргументов соответствующей функции из динамической библиотеки.

## Глава 5

### Установка сервера участником

На данном этапе дипломной работы создан Shell Script для установки Node.js в Ubuntu. В листинге 5.1 представлен Shell Script для установки Node.js и компонентов, необходимых для правильной работы сервера. После скачивания всего проекта Конкурса вызывается это скрипт.

Листинг 5.1. Команды для установки node.js через Терминал

```
1 sudo apt-get install nodejs npm gcc
2 npm install ffi
3 npm install node-ffi
```

Файлы, относящиеся к серверу, а также динамическая библиотека устанавливаются в домашнюю папку каталога.

# Глава 6

## Заключение

### 6.1. Достигнутые результаты

В процессе работы над дипломным проектом:

- Изучено программное обеспечение конкурса КИО.
- Проанализированы достоинства и недостатки различных платформ, которые можно было использовать для реализации сервера. В итоге выбран Node.js и обосновано его использование;
- Спроектирована общая структура сервера, выбраны протоколы обмена между частями системы;
- Для вызова функций из динамической библиотеки используется модуль node-ffi;
- Описана процедура установки и настройки node-ffi под Linux;
- Под Windows установить этот модуль не удалось. Попытки описаны в 3.1.2.
- Реализован сервер на Javascript, который принимает запросы, вызывает нужные функции и возвращает результат;
- Реализован пример на Flash, который демонстрирует соединение приложения с динамической библиотекой через созданный сервер.
- Код протестирован и отвечает поставленном в техническом задании требованиям. Код прокомментирован и документирован в этой пояснительной записке.

- Создан Shell Script для установки node.js для Ubuntu. При сборке всего установщика конкурса при выполнении данного скрипта будет установлена вся серверная часть.

## 6.2. Дальнейшее развитие проекта

- Реализация сервера под Windows (создание скомпилированных версий node-ffi под каждую версию Windows);
- Обработка ошибок при создании запроса программистом, создающим Flash-приложение для (удобства отладки программистом).
- Создание утилиты, которая извлекает сведения о функциях из динамической библиотеки и приводит данные к формату base.txt.

## Глава 7

### Используемая литература

При создании дипломной работы была использована следующая литература, сайты, видео:

1. Книжку про Adobe Flex добавить кто и что
2. Документация Node.js: <http://nodejs.org/api/>
3. Подробный учебник по Node.js: <http://nodebeginner.ru/>;
4. Видео Ryan'a Dahl'a: Знакомство с Node.js:  
[http://www.youtube.com/watch?v=jo\\_B4LTHi3I](http://www.youtube.com/watch?v=jo_B4LTHi3I);
5. Политика безопасности в Flash Player:  
<http://kharchuk.ru/as3/Security.html>;
6. Создание динамической библиотеки в Linux:  
<http://www.firststeps.ru/linux/general1.html>;
7. Документации пакетов LaTeX disser, listings:  
<http://ru.wikipedia.org/wiki/LaTeX> а также примеры внутри пакетов.
8. Книга по Javascript
9. Методичка по оформлению диплома