МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ
РЕСПУБЛИКИКАЗАХСТАН

Некоммерческое акционерное общество

АЛМАТИНСКИЙ УНИВЕРСИТЕТ ЭНЕРГЕТИКИ И СВЯЗИ

ФАКУЛЬТЕТ «АЭРОКОСМИЧЕСКИЕ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

зав. кафедрой «Компьютерные технологии»
д. ф.-м. н., профессор _____ Куралбаев З.К.
«___»_____20 __г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ

пояснительная записка

Технологии разработки распределенных автоматизированных систем управления техническими устройствами

специальность: 6М070400 - Вычислительная техника и программноеобеспечение

Выполнила магистрантка гр. МВТп-13 _____ Ивонина Н.Н.

Научный руководитель к.ф.-м.н., доцент_____Турганбаев Е.С.

Нормоконтролер ст. преп., доктор PhD _____Ержан А.А.

Рецензент д. п. н., профессор _____Сатыбалдиев О.С.

АЛМАТЫ, 2015

СОДЕРЖАНИЕ

АҢДАТПА

Диссертация таратылған есептеу жүйелерін әзірлеу және іске асыру мәселелеріне арналған. Бұл бағыт тиісті және ерте болды, бірақ IPv6 желісін іске қосу және бірінші бұлтты қызметтерді дамыту кейін, ірі корпорациялар қарқынды бөлінген жүйелерге жылжи бастады. Осы жүйелердің модульдік (Service Oriented Platform, SOA) қызмет бағдарланған тұғырнамасының дамуы бастау үшін рұқсат, сондай-ақ қызмет көрсету алаңдары (Service Delivery Platform, SDP), біз Интернетте ең көп сайттарында көре аласыз, ол нәтижелері болды.

Бұл қызметтердің құрылымдық ұйым кез келген ашық терезелер пайдаланушыдан параллель қолдану көптеген қосымша мүмкіндіктер қол жетімді мүмкіндік береді. Бұл жағдайда, сұрау серверде асинхронды өңдеу ақпарат жүйенің жалпы өнімділігі айтарлықтай әсер етпейді.

Бұл мақалада, мен бүгінгі күні сұранысқа ең болып, оларды пайдалану екі платформалар, мысалында таратылған есептеу жүйелерін ұйымдастыру және жұмыс істеу негізгі аспектілерін тоқталсам.

АННОТАЦИЯ

Диссертационная работа посвящена рассмотрению проблемы развития и внедрения распределённых вычислительных систем. Данное направление было актуальным и раннее, однако после начала развития сети IPv6 и разработки первых облачных сервисов, крупные корпорации начали интенсивно переходить на распределённые системы. Модульность таких систем позволило положить начало развитию сервисно-ориентированных платформ (Service Oriented Platform, SOA), а так же платформ предоставления услуг(Service Delivery Platform, SDP), результаты работы которых мы можем наблюдать на большинстве сайтов в Интернете.

Структурная организация подобных сервисов позволяет параллельно использовать множество дополнительных функций, доступных из любого открытого пользователем окна. При этом выполнение запроса не будет оказывать существенного влияния на общую производительность системы благодаря асинхронности обработки информации на серверах.

В данной работе я хочу осветить основные аспекты организации и работы распределённых вычислительных систем на примере двух платформ, использование которых является наиболее востребованным в сегодняшние дни.

ABSTRACT

Dissertational work is devoted to the problems of development and implementation of distributed computing systems. This direction was relevant and early, but after the start of the IPv6 network and the development of the first cloud services, large corporations began to intensively move to distributed systems. The modularity of these systems has allowed to initiate the development of a service-oriented platform (Service Oriented Platform, SOA), as well as service delivery platforms (Service Delivery Platform, SDP), the results of which we can see on most sites on the Internet.

The structural organization of these services allows parallel use many additional features available from any open windows user. In this case, the query will not have a significant impact on the overall system performance by asynchronous processing information on the server.

In this paper, I want to highlight the main aspects of the organization and operation of distributed computing systems on the example of the two platforms, the use of which is the most in demand in today's day.

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

В настоящей магистерской диссертации применяют следующие термины с соответствующими определениями:

SOA – Service-Oriented Architecture, сервисно-ориентированная архитектура

SDP - Service Delivery Platform, платформа предоставления услуг

IDE – Integrated Development Environment, интегрированная среда разработки

OSS - Operation Support System

BSS - Business Support System

HTTP - HyperText Transfer Protocol

HTTPS - Hypertext Transfer Protocol Secure

CORBA – Common Object Request Broker Architecture

SOAP - Simple Object Access Protocol (XML protocol)

IMS - IP Multimedia Subsystem

CMF - Content Management Framework

OpenCV - Open Source Computer Vision Library

Лицензия BSD - Berkley Software Distribution license , программная лицензия университета Беркли

MFC - Microsoft Foundation Classes

Intel IPP - Intel Integrated Performance Primitives

REST - Representational State Transfer , передача репрезентативного состояния

PWM - Pulse Width Modulation, широтно-импульсная модуляция

3dof – 3 dimensions of freedom, три степени свободы

ВВЕДЕНИЕ

Общая характеристика работы. Люди пользуются облачными сервисами, которые позволяют хранить и обрабатывать гигабайты информации на своих серверах, делая их доступными из любой точки земного шара. Структурная организация подобных сервисов позволяет параллельно использовать множество дополнительных функций, доступных из любого открытого пользователем окна. При этом выполнение запроса не будет оказывать существенного влияния на общую производительность системы благодаря асинхронности обработки информации на серверах.

В магистерской диссертации представлена концепция реализации принципов SOA и SDP, продемонстрированы способы взаимодействия их модулей между собой, а так же описаны программные и аппаратные рекомендации, которые следует учитывать при разработке сложных распределённых вычислительных систем.

Постановка проблемы и её актуальность. На сегодняшний день в области IT активно развиваются и внедряются распределённые вычислительные системы. Данное направление было актуальным и раннее, однако после начала развития сети IPv6 и разработки первых облачных сервисов, крупные корпорации начали интенсивно переходить на распределённые системы. Модульность таких систем позволило положить начало развитию сервисно-ориентированных архитектур (Service Oriented Architecture, SOA), а так же платформ предоставления услуг(Service Delivery Platform, SDP), результаты работы которых мы можем наблюдать на большинстве сайтов в Интернете (виджеты «Звонок с сайта», «Консультант online» и другие). Согласно аналитическим данным, в будущем наиболее динамично будут развиваться те сервисы, которые полностью отвечают потребностям современных компаний. Они должны быстро и просто внедряться, не требовать больших инвестиций, иметь постоянную доступность, отсутствие привязанности к географии или к конкретному офисному помещению, иметь лёгкую масштабируемость в случае, если проект или компания расширяются, нанимают внештатных сотрудников или наоборот, сворачиваются по каким-либо причинам.

Необходимо определить инфраструктуру системы, которую предполагается разработать в целях исследования. При этом система должна обеспечивать сочетание нескольких важных характеристик:

- Управляемость и автоматизация;
- Поддержка и сопровождение;
- Масштабируемость;
- Надёжность;
- Высокая готовность;
- Работоспособность.

Целью работы является сравнение возможностей систем, основанных на концепциях SOA и SDP путём их объединения. В соответствии с целью в исследовании следует решить следующие задачи:

1. Исследовать требования и ожидания от систем SOA/SDP.
2. Рассмотреть существующие программные комплексы и их возможности.
3. Оценить требования к программной и аппаратной части.
4. Тестирование и анализ результатов взаимодействия с системой.

Объектом исследования является технология построения и взаимодействия составляющих системы между собой.

Методы исследования. За время работы над диссертацией было изучено множество материалов, представленных на специализированных интернет-порталах. Все базовые понятия и теория были выделены из литературы, указанной авторами статей, в том числе представленные в трудах зарубежных научных исследователей.

Необходимые практические навыки исследования были получены во время прохождения обязательной научной стажировки.

Над аппаратной частью системы проводились эксперименты путём повышения нагрузок на техническую составляющую, программная часть на первых этапах разработки эмулировалась посредством виртуализации. В дальнейшем поведение системы исследовалось после объединения всех частей.

Научную новизну представляет исследование возможности объединения систем, основанных на концепциях SOA и SDP, а так же выведение полученного результата на аппаратный уровень, используя в качестве базы миниатюрный компьютер Raspberry Pi Model B и плату Red Back Spider Robot Controller. В связи с тем, что на сегодняшний день в Казахстане крайне слабо развита отрасль изучения и разработки подобных систем, данное решение позволяет организовать необходимую автоматизацию и масштабируемость разрабатываемой системы, тем самым открывая новые возможности в будущем.

Практическая значимость результатов исследований представляет собой возможность дальнейшего активного развития данного направления. По результатам опросов, на сегодняшний день в Казахстане нет компаний, занимающихся разработкой подобных сложных распределённых вычислительных систем, в связи с чем все основные исследования проходят только в рамках университетов.

Результаты проведённого исследования позволят выявить востребованную и необходимую для внедрения технологию разработки распределённых вычислительных систем. Гибкость и универсальность полученного в итоге комплекса предоставляет право пользования любой комфортной для программиста IDE, что является огромным плюсом при разработке систем, базированных на концепциях SOA.

Личный вклад автора присутствует во всех звеньях работы, а именно: в проведении литературного обзора по теме диссертации, постановке

проблемы, постановке и проведении экспериментов, разработку сложной распределённой вычислительной системы на базе компьютера с минимальными системными требованиями, анализе результатов решения задач, оформлении диссертации.

Апробация

Ивонина Н. Методы организации и разработки распределённых вычислительных систем//Сборник научных трудов магистрантов специальностей «Вычислительная техника и программное обеспечение» и «Информационные системы». – Алматы: АУЭС, 2014 – С.9-14.

Структура и объем диссертации. Диссертационная работа содержит список обозначений и сокращений, введение, основная часть из пяти разделов, заключения, приложения и списка использованных источников. Объем диссертации составляет 73 страницы машинописи, включая 25рисунков, 3 формулы.

1 ОСНОВНЫЕ ПОНЯТИЯ РАСПРЕДЕЛЁННЫХ ВЫЧИСЛИТЕЛЬНЫХ СИСТЕМ

1.1 Основы построения распределённых вычислительных систем

Распределённые вычисления – способ решения трудоёмких вычислительных задач с использованием нескольких компьютеров, чаще всего объединённых в параллельную вычислительную систему. Распределённые вычисления применимы также в распределенных системах управления.

Последовательные вычисления в распределённых системах выполняются с учётом одновременного решения многих задач. Особенностью распределенных многопроцессорных вычислительных систем, в отличие от локальных суперкомпьютеров, является возможность неограниченного наращивания производительности за счет масштабирования. Слабосвязанные, гетерогенные вычислительные системы с высокой степенью распределения выделяют в отдельный класс распределенных систем — грид.

Интеграция информационных и вычислительных ресурсов в единую среду и организация эффективного доступа к ним является одним из основных направлений развития современных информационных технологий. В настоящее время компьютеры, объединенные локальной кооперативной и глобальной сетью, в основном используются как источники информации. Поэтому на первый план выходит проблема эффективного использования вычислительных ресурсов каждой станции сети для решения крупных научных, производственных и технологических задач. Такая постановка вопроса требует создания иной технологии интеграции процессоров, чем технология Internet. Эти технологии в настоящее время развиваются в виде GRID-технологии и технологии «Облако» [1]. Однако подход в предлагаемых системах ориентирован в основном на глобальные вычислительные сети, в то время как ниша корпоративных и кампусных локальных вычислительных сетей, очевидно, требует несколько иного подхода решения задачи интеграции вычислительных ресурсов. Анализ задач, решаемых в корпоративных и производственных сетях, показывает, что эта интеграция должна удовлетворять следующему набору требований:

1. Не требовать использования дополнительного дорогостоящего коммутационного оборудования.
2. Быть достаточно простой в настройке и эксплуатации; не требовать для постоянного обслуживания высококлассных специалистов.
3. Быть «прозрачной» для конечного пользователя, т.е. должна скрывать от него все тонкости ее функционирования.
4. Быть «самонастраиваемой», т.е. уметь поддерживать себя в активном состоянии в любых ситуациях (за исключением форс-мажорных) без вмешательства человека.
5. Поддерживать выполнение задач широкого класса.

6. Быть надежно защищенной от вторжения извне.

7. Предоставлять гибкую систему настройки полномочий каждого из участников вычислительной среды. [2]

Естественно, что эти требования зачастую противоречивы, но именно максимальное удовлетворение им позволит для решения производственных, технологических и офисных задач организации предприятий максимально использовать ресурсы имеющихся компьютеров. В настоящей работе предлагается одна из возможных реализаций такой интеграции для небольших и средних локальных вычислительных сетей.

В настоящее время традиционно сложилось несколько механизмов и архитектур организации общей информационно-вычислительной среды. Наибольшее распространение получила организация клиент–сервер, в которой задания или сетевая нагрузка распределены между поставщиками услуг (сервисов), называемыми серверами, и заказчиками услуг, называемыми клиентами. Такая архитектура позволяет распределить функции вычислительной системы между несколькими независимыми компьютерами в сети, хранить все данные на сервере, защищенные от несанкционированного доступа, объединять различных клиентов для решения одной задачи.

Однако существенным недостатком такой архитектуры является асимметрия системы, в результате которой нагрузка на сервер всегда выше, чем на остальные процессоры сети. Кроме того, эксплуатация системы требует наличия высококвалифицированных специалистов – системных администраторов, обеспечивающих эффективную работоспособность [3].

Более развитой является технология CORBA, основанная на архитектуре брокера объектных запросов. Здесь предусмотрены равноправие всех объектов сети и довольно простые интерфейсы обмена между процессами. Однако в настоящее время эта архитектура существует лишь в спецификациях либо в виде реализации отдельных закрытых подсистем, и пока широкого распространения не получила. Вместе с тем стоит отметить основные ее достоинства: платформенная независимость, языковая независимость, динамические вызовы, динамическое обнаружение объектов, масштабируемость, CORBA-сервисы, широкая индустриальная поддержка [3, 4].

Промежуточное положение между этими технологиями занимает технологический стандарт COM/DCOM, предназначенный для создания программного обеспечения на основе взаимодействующих распределённых компонентов, каждый из которых может использоваться во многих программах одновременно. Технология COM/DCOM основана на интеграции API приложений Windows, что одновременно является ее достоинством и недостатком, так как вне ОС Windows трудно реализуема. Другим недостатком является наличие межсетевых экранов, которые достаточно сложны в настройке [5].

Вместе с тем, на основе COM/DCOM-технологии разработан ряд эффективных программных продуктов, среди которых следует отметить DirectShow. Подобная технология была принята в разработке «Базис-1», «Базис-2», «Базис-3» – систем проектирования, реализации и администрирования распределенных вычислительных систем для различных приложений. В среде «Базис» были реализованы ряд практических задач, в частности: система безопасности «Лик» и система автоматизированного управления экспериментальными исследованиями на бигармоническом лазерном спектрометре. Однако практика использования системы «Базис» для динамически изменяющихся задач показала, что основным недостатком является использование конфигуратора, который настраивается вручную при запуске задачи, и может также вручную меняться в процессе ее решения. Ручная настройка изменения явно вступает в противоречие с требованием оперативности решения тех или иных прикладных задач. В настоящей работе предлагается вариант построения распределенных вычислительных систем для локальных и корпоративных сетей, в котором устранен данный недостаток.

На каждом компьютере вычислительной системы имеется набор модулей, реализующих определенные алгоритмы программного комплекса, например: модуль вычисления обратной матрицы, модуль нахождения градиента изображения. Набор модулей на каждом из узлов сети может отличаться. Каждый из этих модулей может использовать другие для решения подзадач. Например, модуль нахождения обратной матрицы использует модуль решения системы линейных алгебраических уравнений. Система управления распределенными вычислениями включает набор диспетчеров на каждом компьютере сети, выполняющий планирование вычислительного процесса, осуществляющий сборку информации о параметрах и состояния компьютера, ведущий сбор статистической информации и реагирующий на сообщения от диспетчеров других компьютеров сети. Процесс функционирования системы можно разбить на два этапа. [6]

Первый этап представляет собой установку и настройку системы. На этом этапе происходит автоматический сбор параметров сети, которые будут использоваться в дальнейшем при планировании вычислительных процессов. Также на основе полученных данных о параметрах вычислительной сети происходит распределение алгоритмических модулей по узлам сети. Это распределение зависит от наличия свободного дискового пространства, вычислительных возможностей узлов, пропускной способности сети.

Второй этап – использование системы для решения непосредственных вычислительных задач. При запуске вычислительной задачи диспетчер компьютера, на котором эта задача была запущена, планирует процесс выполнения этой задачи таким образом, чтобы процесс максимально соответствовал выбранному критерию оптимальности. Примерами таких критериев могут быть минимальное время выполнения задачи, поддержка

равномерной загрузки всех узлов сети и т.д. Планирование происходит на основе параметров сети, статистической информации, текущей загрузки каждого компьютера и каналов передачи данных. Диспетчер компьютера, на котором была запущена задача, следит за выполнением задачи: собирает статистическую информацию о времени выполнения той или иной подзадачи на конкретном компьютере, принимает решение об изменении плана в критических ситуациях, когда выполнение текущего плана невозможно или нецелесообразно в связи с возникшими непредвиденными обстоятельствами. Такими обстоятельствами могут быть отказ канала передачи данных, резкое повышение загруженности одного узла из-за какой-то другой задачи и т.д. По запросу пользователя выдаются рекомендации об изменении пропускной способности вычислительной сети. Примером рекомендации может служить совет улучшения канала связи в ситуации, когда компьютер с высокой вычислительной способностью находится за узким каналом передачи данных, что делает невозможным его полноценное использование. Такие рекомендации вычисляются на основе статистических данных, собранных в процессе функционирования системы.

Периодически на основе статистических данных или в случае изменения параметров сети система производит перераспределение алгоритмических модулей. Например, в процессе функционирования системы может обнаружиться, что если бы на определенном узле был определенный алгоритмический модуль, то множество вычислительных задач удалось бы завершить быстрее.

1.1.1 Значение SOA (Service Oriented Architecture)

В дальнейшем на основе понятий распределённых вычислительных систем была разработана сервисно-ориентированная архитектура(Service-Oriented Architecture, SOA). По словам Клива Финкельштейна [9], до появления концепции SOA при разработке систем в качестве отправного момента для программирования бизнес-логики использовались диаграммы рабочих потоков и блок-схемы систем. Разработанные вручную программы тщательно тестировались, после чего внедрялись. Сегодня ситуация изменилась коренным образом: современные инструменты управления бизнес-процессами позволяют обойтись без ручной разработки и тестировании. Так, с помощью методов моделирования можно проверять корректность исполнения бизнес-логики, представленной в диаграммах, а затем автоматически получать описания этих диаграмм на XML-языках управления бизнес-процессами.

По мнению Клива Финкельштейна и Джереми Уэстермана [7-9], такая технология управления бизнес-процессами является большим шагом вперед с точки зрения повышения эффективности разработки систем; по значимости ее можно сравнить с созданием в конце 50-х годов компиляторов языка высокого уровня. Действительно, данный подход позволяет упростить вызов

Web-сервисов из любого местоположения и их выполнение на основе бизнес-правил. Кроме того, при изменении этих правил, корректируется соответствующая логика в диаграммах: диаграммы автоматически генерируются заново. Таким образом, закладываются предпосылки для перехода от медленного ручного кодирования, используемого сейчас при создании систем, к автоматизированному. Благодаря этому компании смогут реализовывать изменение бизнес-правил за минуты или часы, а не за месяцы или годы.

В одной из своих статей, посвященных SOA, Джереми Уэстерман описывает основные на проблемные места и приводит подробные пояснения:

1. SOA не является чем-то новым: IT-отделы компаний успешно создавали и развертывали приложения, поддерживающие сервис-ориентированную архитектуру уже много лет - задолго до появления XML и Web-сервисов.
2. SOA - это не технология, а способ проектирования и организации информационной архитектуры и бизнес-функциональности.
3. Покупка самых новых продуктов, реализующих XML и Web-сервисы, не означает построения приложений в соответствии с принципами SOA.

В некоторых источниках[10] дается следующее определение SOA: это целая философия проектирования, разработки и управления дискретных единиц логики сервисов в вычислительной среде. Применение этого подхода требует от разработчиков проектирования приложений в качестве набора сервисов, даже если преимущества такого решения сразу неочевидны. Разработчикам предоставляется возможность воспользоваться уже существующими сервисами, или изучить, как их сервисы могут быть использованы их коллегами.

SOA "подталкивает" к использованию альтернативных технологий и подходов (таких как обмен сообщениями) для построения приложений посредством связывания сервисов, а не посредством написания нового программного кода. В этом случае, при надлежащем проектировании, применение сообщений позволяет компаниям своевременно реагировать на изменение рыночных условий - "настраивать" процесс обмена сообщениями, а не разрабатывать новые приложения.

Однако не все источники сходятся в едином мнении в попытках дать точное определение SOA. Некоторые определяют SOA как термин, который появился для описания исполняемых компонентов - таких как Web-сервисы - которые могут вызываться другими программами, выступающими в качестве клиентов или потребителей этих сервисов. Эти сервисы могут быть полностью современными - или даже устаревшими - прикладными программами, которые можно активизировать как черный ящик. От разработчика не требуется знать, как работает программа, необходимо лишь понимать, какие входные и выходные данных нужны, и как вызываются эти программы для исполнения. [11]

В самом общем виде SOA предполагает наличие трех основных участников: поставщика сервиса, потребителя сервиса и реестра сервисов (см. рис. 2). Взаимодействие участников выглядит достаточно просто: поставщик сервиса регистрирует свои сервисы в реестре, а потребитель обращается к реестру с запросом.

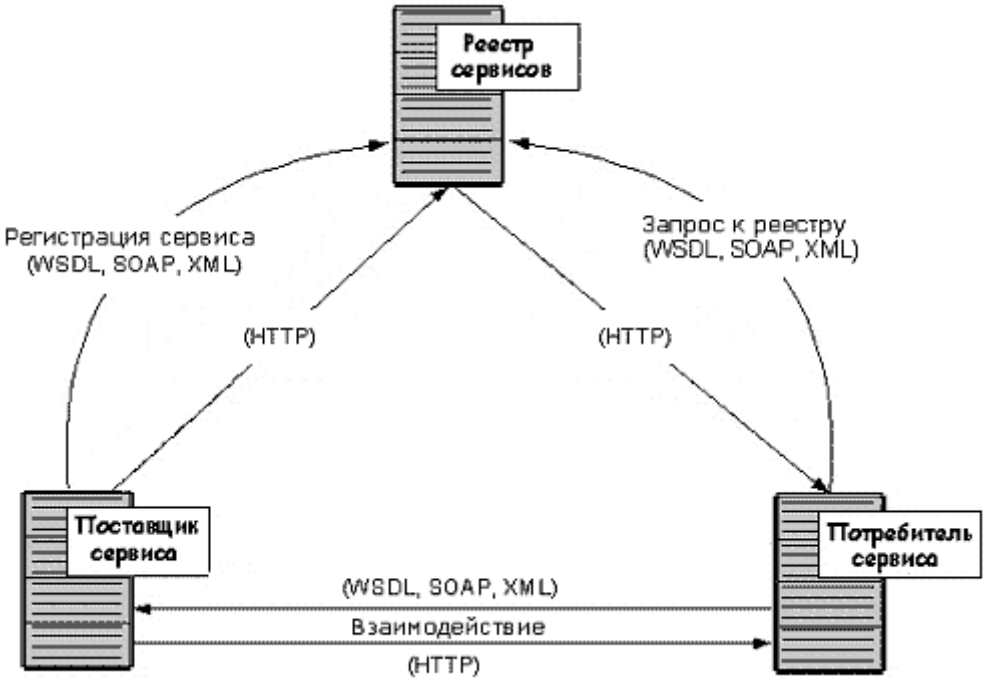

Рисунок 2 - Общая схема SOA

Для использования сервиса необходимо следовать соглашению об интерфейсе для обращения к сервису - интерфейс должен не зависеть от платформы. SOA реализует масштабируемость сервисов - возможность добавления сервисов, а также их модернизацию. Поставщик сервиса и его потребитель оказываются несвязанными - они общаются с помощью сообщений. Поскольку интерфейс должен не зависеть от платформы, то и технология, используемая для определения сообщений, также должна не зависеть от платформы. Поэтому, как правило, сообщения являются XML-документами, которые соответствуют XML-схеме.[5]

Любой разговор о SOA невольно переходит на рассуждение о роли и месте Web-сервисов. Несмотря на то, что основные положения SOA сложились задолго до появления Web-сервисов, сегодня Web-сервисы занимают центральное место в SOA. Так, Джереми Уэстерман отмечает [6-9], что использование XML и Web-сервисов "поднимает SOA на более высокий уровень".

Действительно, открытые стандарты, описывающие XML и Web-сервисы, позволяют применять SOA ко всем технологиям и приложениям, установленным в компании. Как известно, Web-сервисы базируются на широко распространенных и открытых протоколах: HTTP, XML, UDDI,

WSDL и SOAP. Именно эти стандарты реализуют основные требования SOA - во-первых, сервис должен поддаваться динамическому обнаружению и вызову (UDDI, WSDL и SOAP), во-вторых, должен использоваться независящий от платформы интерфейс (XML). Наконец, HTTP обеспечивает функциональную совместимость.

Однако прежде чем перечислить преимущества использования SOA, будет уместным напомнить, что преимущества бывают разные: стратегические и тактические. SOA обладает рядом достоинств как стратегических, так и тактических.

Стратегическая ценность SOA:

- Сокращение времени реализации проектов, или "времени выхода на рынок".
- Повышение производительности.
- Более быстрая и менее дорогая интеграция приложений и интеграция B2B - остановимся более подробно на данном пункте.

Известно, что реализация традиционных решений для интеграции прикладных программ - непростая задача, требующая существенных капиталовложений. Кроме того, часто при внедрение необходимо написание программного кода. SOA предусматривает размещение сервисов в сети в режиме исполнения, т.е. позволяет автоматизировать эти ресурсоемкие процессы, благодаря чему существенно сокращаются все расходы на интеграцию[10-12].

Тактические преимущества SOA:

- Более простые разработка и внедрение приложений.
- Использование текущих инвестиций.
- Уменьшение риска, связанного с внедрением проектов в области автоматизацией услуг и процессов.
- Возможность непрерывного улучшения предоставляемой услуги.
- Сокращение числа обращений за технической поддержкой.
- Повышение показателя возврата инвестиций (ROI).

Согласно исследованиям, результаты которых приведены в работе [13] можно выделить в качестве основной проблемы SOA проектирование интерфейса, что, в свою очередь, выдвигает новые требования к разработчикам, связанные с кардинальной переменой самой идеологии программной разработки, которые не могут обеспечить современные средства проектирования, изначально ориентированные на классическую клиент-серверную архитектуру. Также в связи с несогласованностью стандартов в области Web-сервисов сложились две группировки: одна включает IBM, BEA и Microsoft, а вторая — Sun, Fujitsu и Oracle. Каждая из них продвигает свои разработки. Например, для управления транзакциями первая предлагает протокол WS-Transactions, а вторая — WS-Transactions Management; для гарантированной доставки сообщений первая выпустила

WS-ReliableMessaging, а вторая — WS-Reliability. (Стоит отметить, что после анонсирования Microsoft Visual Studio Team System 2005, а также предложений IBM по конкретным решениям, основанным на SOA, это противостояние стало минимальным, и можно ожидать продвижения в разработке единых стандартов, регламентирующих использование Web-сервисов)

1.1.2 Значение SDP (Service Delivery Platform)

Service Delivery Platform (SDP) - платформа предоставления услуг, в телекоммуникации обычно представляет собой набор компонентов, которые управляют процессами создания сервисов (услуг), а также позволяют управлять сессиями и протоколами различного телекоммуникационного оборудования предприятия. На текущий момент единого стандарта определения в индустрии понятия SDP не существует, и каждый вендор, предлагающий некую платформу, определяет состав и наличие компонентов системы различными способами. [14]

Сервисная платформа обычно требует интеграции телекоммуникационного оборудования и информационных технологий, с возможностью создания служб, сервисов и услуг, которые выходят за рамки ограничения IT технологии и сетевой инфраструктуры. Данное решение, как правило, оптимизировано для создания IMS сервисов (передача мультимедийного содержимого по IP), IPTV, мобильное ТВ и т.д. В состав платформы, как правило, входит среда разработки, с необходимым набором инструментов для создания, управления и контроля работы сервисов (услуг), оркестровки, и непосредственное исполнение процессов на платформе. Также предоставляемые услуги могут включать в свой состав механизмы присутствия, использующиеся в службах мгновенного обмена сообщениями (IM), местоположения (Geolocation), телеметрию (Telemetrics) и другие возможности низкоуровневых коммуникаций. Платформа SDP может быть применима как для конечных потребителей услуг, так и для работы бизнес-приложений.

Бизнес-целью реализации платформы SDP является возможность быстрой разработки и развертывания новых конвергентных мультимедийных услуг, начиная от основных услуг телефонии до сложных аудио/видео конференций в многопользовательских сервисах развлечений (например, игры MPGs). Используя сетевые ресурсы платформ SDP можно управлять жизненным циклом тысяч приложений, а также осуществлять взаиморасчеты с разработчиками данных приложений и потребителями услуг. Появляется возможность выхода на рынок широкого спектра различных приложений и услуг, например:

Пользователи получают информацию о входящих телефонных вызовах (как проводных, так и мобильных устройств), сообщения из социальных

сетей, интернет пейджеров, или данные о местоположении друзей (поддержка GPS/GSM), прямо на экран телевизора.

Пользователи могут заказать услуги VoD (видео по запросу) для мобильных телефонов или смотреть потоковое видео. Также появилась возможность заказать видео пакет для обоих устройств, на мобильный телефон или домашний телевизор.

Клиенты авиакомпаний получают текстовые сообщения (SMS) от автоматизированных систем, относительно отмены рейса, с возможность дальнейшего использования меню IVR или мобильного приложения для самообслуживания и перепланирования рейса.

Основным критерием при выборе сервисной платформы является поддержка широкого спектра телекоммуникационного оборудования, протоколов связи, быстрота разработки и внедрения новых услуг, при минимальных накладных расходах, включая техническую поддержку и сопровождение самой платформы, а также отсутствие или минимальное количество ПО третьих сторон.

Согласно мировому тренду растет не только суммарный объем доходов от услуг, но и спектр предлагаемых услуг, при этом, появляется множество нишевых услуг и услуг с коротким жизненным циклом. Большая часть рынка нишевых услуг и услуг с коротким жизненным циклом в настоящее время приходится на сервис-провайдеров. Ограничивающие факторы, накладываемые организационной структурой, не позволяют оператору оперативно перейти к предоставлению услуг в соответствии с новыми бизнес-моделями, и сократить себестоимость разработки и сроки вывода услуг на рынок. А при использовании старых технологий создания новых услуг, доходы от услуги с коротким жизненным циклом не позволяют окупить значительные затраты на их разработку.

Описанные выше процессы обуславливают постепенный переход рынка телекоммуникационных услуг к построению «бизнес-инкубаторов» – модели, при которой операторы связи и сервис-провайдеры предоставляют не только своим, но и внешним разработчикам все необходимые инструменты для быстрого построения самостоятельного или совместного бизнеса предоставления услуг:

- средства создания услуг с интерфейсом к готовым сервисам оператора связи;
- модельную сеть для отладки и тестирования;
- средства контроля и управления услугами;
- средства тарификации услуг;
- средства быстрого внедрения услуг на рынок – каналы продаж и средства проведения рекламных кампаний;
- службу технической поддержки – контакт центры первой и второй линии, интегрированные со средствами создания, управления и мониторинга услуг.

Модель «бизнес-инкубатор» позволяет существенно снизить риски, связанные с выводом нового продукта на рынок. Помимо создания собственных услуг, операторы связи могут отбирать удачные решения и инвестировать средства в успешные услуги внешних разработчиков.

Модель SDP предполагает, что создание, предоставление, учет и тарификация услуг осуществляется посредством ресурсов оператора связи. Продажа услуг осуществляется через все имеющиеся у оператора каналы продаж и через специализированный портал. Соответственно, вся выручка поступает на счет оператора, и разделяется с партнерами. Продвижением услуг занимаются партнеры.

Сервисно-ориентированный подход с применением Service Oriented Architecture (SOA) к построению архитектуры сервисного слоя, позволяющего максимально ускорить разработку и внедрение новых услуг, заключается в максимальном использовании существующих сервисов и задействовании их в качестве составных элементов новых более сложных услуг для абонентов. [14, с.56]

Платформа доставки услуг SDP для реализации сервисного слоя на базе подхода SOA становится централизованной платформой для разработки, развертывания и предоставления услуг конечным пользователям.

SOA дает возможность отделения жизненного цикла телекоммуникационной услуги от жизненного цикла крупных узлов функциональности, таких как системы OSS/BSS (включая биллинг), CRM, и т.д. Четкое разделение между общими функциями и специфичными только для данной телекоммуникационной услуги.

Используя принципы SOA, выделяются бизнес-сервисы для интегрирования крупных функциональных узлов для предоставления различных услуг потребителям. Это сокращает время запуска услуги за счет существенного упрощения интеграции различных компонентов друг с другом при условии следования концепции SOA: слабая связанность интерфейсов, каталогизация сервисов и т.д. Возможность повторного или многократного использования существующих сервисов как при разработке, так и в процессе эксплуатации, является универсальным преимуществом SOA, что для телекоммуникационных компаний исключительно важно.

Применение подхода SOA на практике, основано на четком соответствии принятым телекоммуникационным и ИТ-стандартам, предполагает независимость решения от конкретной программной платформы или технологий реализации самой телекоммуникационной среды. Таким образом, интегрируются разнородные технологии (Messaging, Streaming, Web и т.п.), позволяя делать все более сложные технически и актуальные предложения для конечных абонентов.

Согласно одной из официально представленных документаций [14] подход к построению SDP заключается в применении сервис-ориентированного подхода SOA с выделением:

- компонентов для поддержки сервисов;

- базовых сервисов;
- составных сервисов;
- специализированных сервисов.

Услуги для конечного абонента представляют собой приложения, состоящие как из поддерживающих компонентов, так и из базовых, составных и специализированных сервисов.

Логика сервиса может быть реализована посредством объединения базовых компонентов различного типа.

Компоненты для поддержки сервисов – поддерживают общие для нескольких сервисов функции, гарантирующие единообразие и снижение расходов при многократном использовании. Как правило, компоненты для поддержки сервисов – это сетевые функции, предлагающие стандартизированные интерфейсы к сервисам. Примеры компонентов этой категории: сервисы местоположения, обмен сообщениями, сервисы присутствия и списки рабочих групп.

Базовые сервисы – простые сервисы, которые включены в состав каталога сервисов оператора и могут быть заказаны абонентами. Базовые сервисы способны вызывать компоненты для поддержки сервисов и могут иметь как стандартизированные, так и интерфейсы собственной разработки. Примеры базовых сервисов: календарные сервисы и голосовая конференц-связь и др.

Составные сервисы – комбинированные сервисы, представленные двумя распространенными типовыми вариантами.

Составной сервис объединяет два (или более) базовых сервиса или компонента поддержки сервисов в рамках интегрированного пакета сервисов с расширенной функциональностью. Например, сервис картографирования, сочетающий сервис местоположения (сведения о местоположении устройства) с сервисом извлечения контента (отображение карты).

Составной сервис представляет собой комплекс различных предложений на базе одного сервиса, что позволяет быстро вывести этот сервис на несколько рынков. Например, один и тот же коммуникационный сервис от стороннего поставщика может быть предложен клиентам на различных финансовых условиях: дебетовая оплата (pre-paid), оплата по факту (post-paid), бесплатное пользование с обязательным рекламным сопровождением и т.д.

Специализированные сервисы – наивысший уровень составных сервисов; вертикально организованные сервисы, которые применяются за пределами телекоммуникационной сети. Некоторые услуги могут быть реализованы посредством специализированных сервисов. Эти сервисы предназначены для использования разработчиками нетелекоммуникационных компаний, которые концентрируют свои усилия на контенте и на специфических функциях своего рыночного сегмента. Примеры: спортивно-развлекательные компании, распространяющие

результаты состязаний и видеоматериалы среди своих подписчиков (телевидение, Интернет, газеты), или телевизионные шоу, использующие дистанционное голосование как элемент своих программ.

Технологии, применяемые для реализации подхода SOA, например технология Web-сервисов, позволяют объединять произвольные компоненты для сборки инновационных сервисов.

SDP поддерживает интеграцию компонентов на базе подхода SOA с целью обеспечения доступа конечных пользователей к составным сервисам. Решение обеспечивает персонализацию сервисов, что упрощает получение требуемых сервисов конечным пользователем наиболее удобным для него способом.

Архитектура SDP предполагает наличие следующих уровней создания и выполнения услуг:

- **Уровень услуг:** На данном уровне располагаются реализации конечных сложных услуг для абонентов. Этот слой использует сервисные платформы и системы OSS/BSS уже развернутые в компании, а также уровни исполнения и оркестрации сервисов и исполнения процессов, входящие в состав SDP, и услуги сторонних сервис-провайдеров.
- **Уровень исполнения процессов:** Элементы этого уровня берут на себя управление бизнес-процессами и интеграцию с существующими у Оператора системами OSS/BSS. Сервисы, вызываемые на этом уровне, могут располагаться на удалённых серверах, бизнес-процессы могут занимать длительное время (часы и дни), поскольку их участниками являются не только информационные системы, но и люди (персонал Оператора и абоненты). Обеспечивается логика выполнения длительных услуг. Например, абонент через портал заказал себе услугу, предоставление которой требует подписания договоров, проверки технической возможности и тому подобных процессов, в которых задействован персонал. Обеспечивается выполнение этих процессов, уведомление персонала, внесение необходимых записей в системы OSS/BSS Оператора и резервируя сетевые ресурсы и сервисы под создаваемую услугу. На этом уровне также обеспечивается каталогизация сервисов, хранятся сведения обо всех имеющихся сервисах, их параметрах, версиях, модификациях и т.п. (см. Рисунок 3).

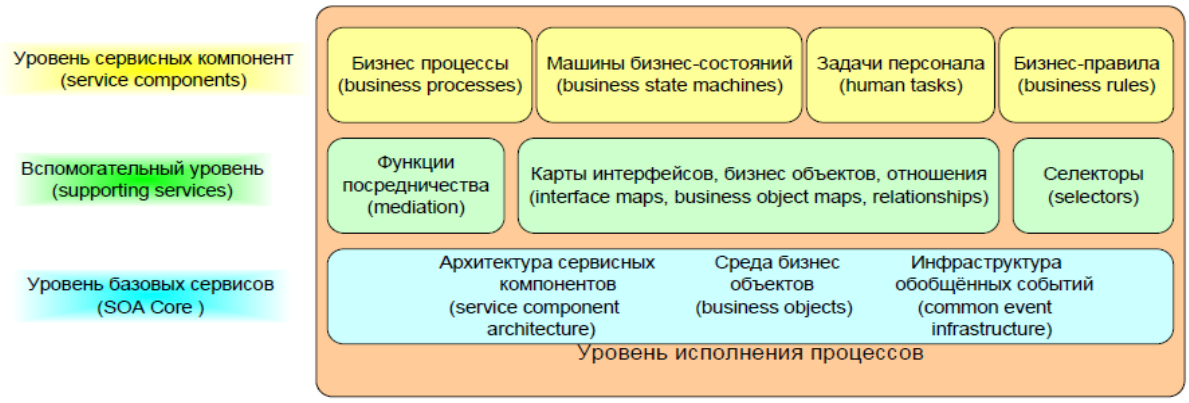

Рисунок 3 – Уровень исполнения процессов

❖ Уровень базовых сервисов (SOA Core): Решает базовые проблемы, связанные с реализацией сервис-ориентированной архитектуры. Состоит из унифицированной программной модели, позволяющей единообразно представлять сервисы без необходимости учёта специфики их реализации. SCA определяет, каким образом сервисы могут вызываться и собираться в композитные приложения.

❖ Вспомогательный уровень: Необходим для согласования интерфейсов сервисов. Во многих случаях интерфейсы существующих компонентов соответствуют друг другу семантически, но имеют различный синтаксис (например, updateCustomer и updateCustomerInDB2). Отношения обеспечивают сопоставление идентификаторов, используемых в различных информационных системах для одних и тех же сущностей. Сценарий бизнес интеграции часто предполагает обращение к одинаковым данным (например, к записям о клиентах), используемым различными серверными системами, например, ERP и CRM.

❖ Уровень сервисных компонентов**:** Реализует бизнес-логику. К ним относятся бизнес-процессы (business processes), бизнес-правила (business rules), Задачи персонала (human tasks) и машины бизнес-состояний

• **Уровень исполнения и оркестрации сервисов:** предназначен для управления вызовом и оркестрации сервисов, время исполнения которых исчисляется секундами (выполняемым в реальном масштабе времени). Элементы этого уровня отвечают за исполнение телекоммуникационных сервисов, которые выполняются в течение краткого промежутка времени. Например, отправка SMS, MMS, установление соединения, индикация информации о присутствии или статусе терминала, передача информации в биллинговую систему и т.д. Они предоставляют доступ к телекоммуникационным сервисам и системам биллинга для сервисов, исполняемых на вышележащих уровнях. Обеспечивается сокрытие структуры сети, контролируется доступ и учет использования ресурсов сети.

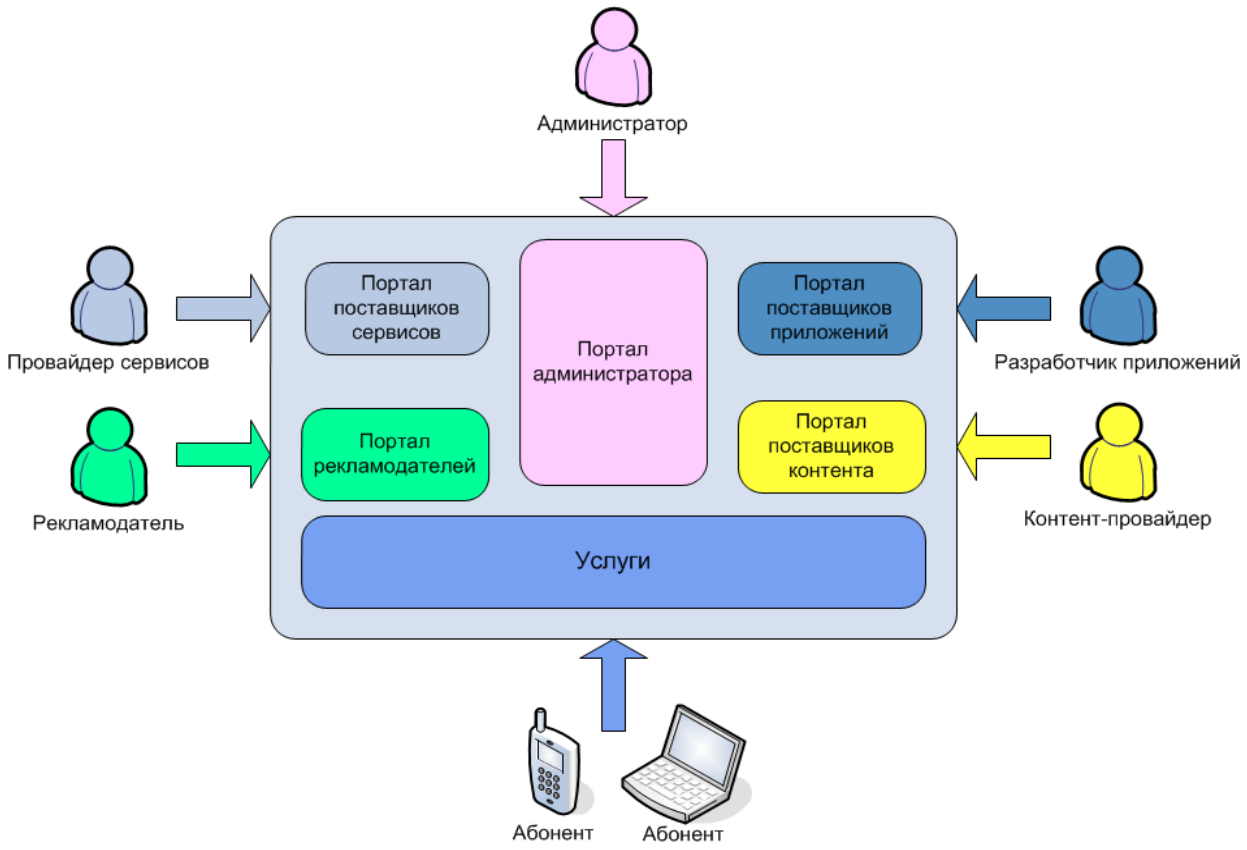

Рисунок 4 – Пример структуры портала в решении SDP

Платформа доставки услуг представляет собой надежное решение промышленного класса. Минимальный уровень отказоустойчивости оборудования в составе платформы 99,999%. В комплексе вся платформа обладает отказоустойчивостью на уровне 99,999%. Все функциональные компоненты, модули и подсистемы платформы доставки услуг поставляются с резервным оборудованием, поддерживающим режим горячего резервирование.

Серверы баз данных, серверы управления сигнализацией и серверы исполнения логики услуг поставляются в кластерных вариантах, работающих в режиме «failover».

Серверы обработки медиа трафика (голос и видео) являются взаимозаменяемыми, поддерживают линейное масштабирование и работают в режиме «load balance». Схема размещения серверов обработки медиа-трафика также обеспечивает их географическое резервирование. Так же следует отметить, что все функциональные модули платформы поддерживают возможность линейного масштабирования. Каждый функциональный модуль может быть умощнен дополнительным аппаратным обеспечением, позволяющим увеличить производительность данного модуля.

К примеру, услуга Виртуальный Call Center (в пакете с многоканальным виртуальным номером) гарантирует, что компания может вести любое количество разговоров одновременно, а правильно настроенный IVR позволяет моментально "доставить" звонок в нужный отдел, без

утомительного перекидывания абонента от одного сотрудника к другому, да и вообще - сделать время ожидания на линии минимальным для любого абонента. Результаты ноябрьского исследования, проведенного компанией "Открытые Коммуникации", выяснявшей качество обслуживания клиентов в call-центрах ведущих купонных сервисов, подтверждают, что пользователь контакт-центра в среднем ждет на линии 11 секунд без IVR и 26 секунд с учетом работы голосового меню.

1.2 Механизмы взаимодействия модулей системы

Как правило, все компоненты любой распределённой вычислительной системы (вне зависимости от сложности) взаимодействуют между собой по заранее заданным и настроенным протоколам. Однако на сегодняшний день, в целях повышения гибкости и быстродействия своих систем всё больше компаний допускают использование фреймворков.

Фреймворк (англ. framework — каркас, структура) — структура программной системы; программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта. Употребляется также слово «каркас», а некоторые авторы используют его в качестве основного, в том числе не базируясь вообще на англоязычном аналоге. Можно также говорить о каркасном подходе как о подходе к построению программ, где любая конфигурация программы строится из двух частей: первая, постоянная часть — каркас, не меняющийся от конфигурации к конфигурации и несущий в себе гнезда, в которых размещается вторая, переменная часть — сменные модули (или точки расширения).

Фреймворк отличается от понятия библиотеки тем, что библиотека может быть использована в программном продукте просто как набор подпрограмм близкой функциональности, не влияя на архитектуру программного продукта и не накладывая на неё никаких ограничений. В то время как каркас диктует правила построения архитектуры приложения, задавая на начальном этапе разработки поведение по умолчанию, каркас, который нужно будет расширять и изменять согласно указанным требованиям. Пример программного фреймворка — CMF (Content Management Framework), а пример библиотеки — модуль электронной почты.

Также, в отличие от библиотеки, которая объединяет в себе набор близкой функциональности, каркас может содержать в себе большое число разных по тематике библиотек.

Другим ключевым отличием фреймворка от библиотеки может быть инверсия управления: пользовательский код вызывает функции библиотеки (или классы) и получает управление после вызова. Во фреймворке пользовательский код может реализовывать конкретное поведение, встраиваемое в более общий, абстрактный код фреймворка. При этом фреймворк вызывает функции (классы) пользовательского кода.

Одним из главных преимуществ при использовании каркасных приложений является то, что такие приложения имеют стандартную структуру. Каркасы приложения стали популярны с появлением графических интерфейсов пользователя, которые имели тенденцию к реализации стандартной структуры для приложений. С их использованием стало гораздо проще создавать средства для автоматического создания графических интерфейсов, так как структура внутренней реализации кода приложения стала известна заранее. Для обеспечения каркаса обычно используются техники объектно-ориентированного программирования (например, части приложения могут наследоваться от базовых классов фреймворка).

Одним из первых коммерческих фреймворков приложения был MacApp, написанный Apple под Macintosh. Первоначально созданный с помощью расширенной (объектно-ориентированной) версии языка Паскаль, впоследствии он был переписан на C++. Другие популярные каркасы для Macintosh включали Metrowerks Powerplant и MacZoop (все основаны на Carbon). Также WebObjects от NeXT.

В различной степени фреймворки приложения представляют собой Cocoa для Mac OS X, а также свободные фреймворки, существующие как часть проектов Mozilla,OpenOffice.org, GNOME и KDE.

Microsoft создала похожий продукт для Windows, который называется «Microsoft Foundation Classes» (MFC). На данный момент основным продуктом Microsoft для разработки ПО предлагается .NET Framework.

Кроссплатформенными каркасами приложений для операционных систем Linux, Macintosh и Windows являются, например, widget toolkit, wxWidgets, Qt, MyCore или FOX toolkit.

Фреймворк определяется как множество конкретных и абстрактных классов, а также определений способов их взаимоотношения. Конкретные классы обычно реализуют взаимные отношения между классами. Абстрактные классы представляют собой точки расширения, в которых каркасы могут быть использованы или адаптированы.

Точка расширения — это та часть фреймворка, для которой не приведена реализация. Соответственно каркас концептуальной модели состоит из концептуальных классов, а каркас программной системы из классов языка программирования общего назначения.

Процесс создания фреймворка заключается в выборе подмножества задач проблемы и их реализаций. В ходе реализаций общие средства решения задач заключаются в конкретных классах, а изменяемые средства выносятся в точки расширения.

Выводы по первой главе

Технологии, применяемые для реализации подхода SOA, например технология Web-сервисов, позволяют объединять произвольные компоненты для сборки инновационных сервисов.

SDP поддерживает интеграцию компонентов на базе подхода SOA с целью обеспечения доступа конечных пользователей к составным сервисам. Решение обеспечивает персонализацию сервисов, что упрощает получение требуемых сервисов конечным пользователем наиболее удобным для него способом.

2 РАЗРАБОТКА И ОБОСНОВАНИЕ ПРОЕКТНО-ПРАКТИЧЕСКИХ РЕКОМЕНДАЦИЙ ПО РЕАЛИЗАЦИИ БАЗОВЫХ ВОЗМОЖНОСТЕЙ РАСПРЕДЕЛЁННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

2.1 Использование и настройка готовых решений для хранения и обработки данных

Так как одна из концепций SOA подразумевает под собой использование не только самостоятельно разработанных программных комплексов, но и готовые решения сторонних разработчиков, за время исследования был проведён подбор и настройка соответствующих фреймворков и библиотек в соответствии с требованиями к производительности вычислительной системы.

В качестве ядра системы был выбран один из дистрибутивов семейства Linux – Raspbian. Дистрибутив основан на пакетной базе Debian Wheezy и специально оптимизирован для Raspberry Pi (сборка для ARMv6 с расширениями "hard float"). Однако при возможности рекомендуется установить Ubuntu версии 14 и выше, и на её основе мигрировать на ROS – Robot Operating System. Это так называемая мета-система, устанавливаемая поверх существующей и предоставляющей доступ к аппаратной части(даже низкоуровневой), имеющая функции для общения между процессами и управления пакетами. Вместе с мета-системой поставляется пакет программного обеспечения, необходимого для разработки программной части, моделирования нагрузок и поведения, а так же управления существующими процессами.

Основным связующим звеном между элементами системы используется фреймворк WebIOPi(рис.5), который позволяет контролировать состояние и управлять всеми портами GPIO локально или удаленно, из браузера или любого приложения.[15] Возможности(рис.7):

- REST API через HTTP и CoAP с поддержкой мультикаста
- Сервер написан на Python
- Работа с GPIO, Serial, 1-Wire
- Совместимость с Python версий 2 и 3
- Великолепные возможности адаптации под нужды пользователей
- Защита логином и паролем

Данный продукт был разработан специально для расширения возможностей и применения Raspberry Pi в автоматизации и робототехнике. Использует REST (сокр. от англ. Representational State Transfer — «передача репрезентативного состояния») — метод взаимодействия компонентов распределённого приложения в сети Интернет, при котором вызов удаленной процедуры представляет собой обычный HTTP-запрос (обычно GET или POST; такой запрос называют REST-запрос), а необходимые данные передаются в качестве параметров запроса. Этот способ является

альтернативой более сложным методам, таким как SOAP,CORBA и RPC(рис.6).

Все GPIO могут быть использованы непосредственно с API REST. Например, чтобы установить GPIO 23 в качестве выхода необходимо осуществить POST HTTP-запрос на /GPIO/23/function/out, что даёт на выходе логическую 1, и отправить POST на /GPIO/23/value/1. Чтобы узнать текущее состояние порта, отправить HTTP-запрос GET на /GPIO/23/function и /GPIO/23/value. [15]

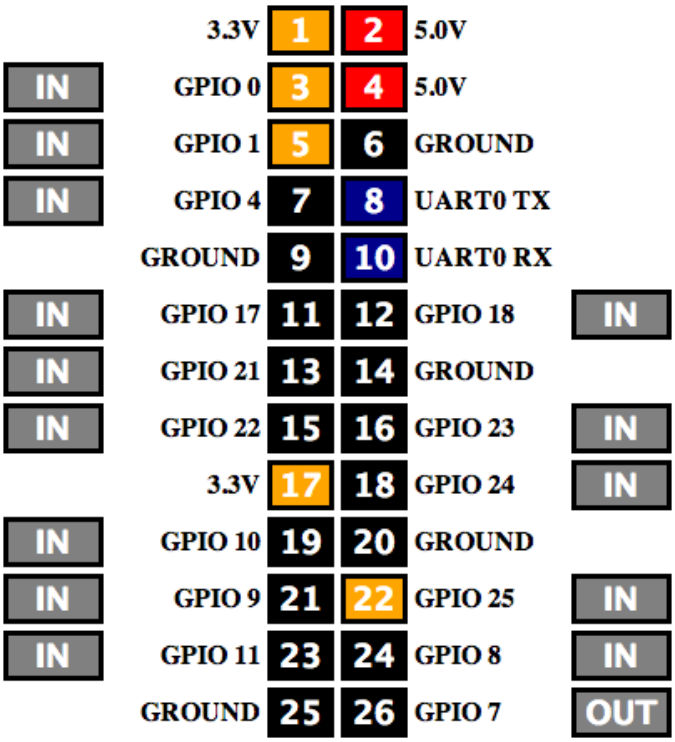

Рисунок 5 – Интерфейс WebIOPi

Использование библиотек Javascript позволяет вносить изменения в состояния GPIO, следить за напряжением на используемых пинах, а так же использовать мониторинг в режиме реального времени не заботясь о вызовах REST.

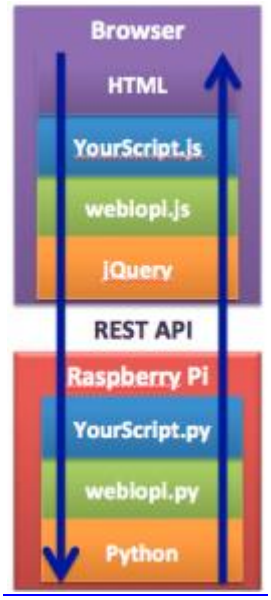

Рисунок 6 – Структура WebIOPi

Рисунок 7 – Возможности WebIOPi

Для установки необходимо, что бы на Raspberry Pi был установлен Python версии 2.7 или 3.2. Инсталляция выполняется четырьмя командами из терминала с правами администратора, локально или удаленно:

```
$ wget http://webiopi.googlecode.com/files/WebIOPi-0.6.0.tar.gz
$ tar xvzf WebIOPi-0.6.0.tar.gz
$ cd WebIOPi-0.6.0
$ sudo ./setup.sh
```

Теперь можно запустить webiopi в командной строке:

```
$ sudo webiopi [-h] [-c config] [-l log] [-s script] [-d] [port]
```

Options:

```
  -h, --help          Display this help
  -c, --config  file   Load config from file
  -l, --log     file   Log to file
  -s, --script  file   Load script from file
  -d, --debug         Enable DEBUG

 Arguments:
  port                Port to bind the HTTP Server
```

Однако после завершения выполнения скрипта или закрытия окна терминала останавливается сервер WebIOPi и происходит сброс состояния портов GPIO. В связи с этим, WebIOPi стоит запускать и останавливать как сервис:

```
$ sudo /etc/init.d/webiopi start
        и
$ sudo /etc/init.d/webiopi stop
```

Работу фреймворка можно проверить в локальной сети, если набрать http://[IP_ Raspberry Pi]:8000 на любом компьютере и получить доступ к web-интерфейсу к Raspberry Pi. При каждом обращении пользователь должен пройти авторизацию. По умолчанию указывается логин «webiopi» и пароль «raspberry».

На сегодняшний день доступны два оптимальных решения для работы с видео: openFrameworks[16] и OpenCV. Оба решения имеют открытый исходный код и находятся в свободном доступе. Так как openFrameworks не оптимизирован для работы на процессорах типа ARMv6(на котором основан Raspberry Pi), этот вариант был изучен в ознакомительных целях. Однако при наличии возможности, в проектах рекомендуется использовать именно этот фреймворк, так как он подразумевает наличие множества удобных для разработки библиотек.

В связи с вышеуказанным, для работы с видеопотоком была выбрана библиотека OpenCV (англ. Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) - набор алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Она реализована на C/C++, также на сегодняшний день разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков. Может свободно использоваться в академических и коммерческих целях — распространяется в условиях лицензии BSD. [17]

Библиотека была разработана для способствования росту числа таких приложений и создания новых моделей использования PC, а так же с целью сделать платформы Intel более привлекательными для разработчиков за счёт

дополнительного ускорения с помощью Intel® Performance Libraries. OpenCV способна автоматически обнаруживать присутствие IPP и MKL и использовать их для ускорения обработки.

В версии 2.2 библиотека была реорганизована. Вместо универсальных модулей cxcore, cvaux, highGUI и других было создано несколько компактных модулей с более узкой специализацией:

- opencv_core — основная функциональность. Включает в себя базовые структуры, вычисления(математические функции, генераторы случайных чисел) и линейную алгебру, DFT, DCT, ввод/вывод для XML и YAWL и т. д.
- opencv_imgproc — обработка изображений (фильтрация, геометрические преобразования, преобразование цветовых пространств и т. д.).
- opencv_highgui — простой UI, ввод/вывод изображений и видео.
- opencv_ml — модели машинного обучения (SVM, деревья решений, обучение со стимулированием и т. д.).
- opencv_features2d — распознавание и описание плоских примитивов (SURF, FAST и другие, включая специализированный фреймворк).
- opencv_video — анализ движения и отслеживание объектов (оптический поток, шаблоны движения, устранение фона).
- opencv_objdetect — обнаружение объектов на изображении (нахождение лиц с помощью алгоритма Виолы-Джонса, распознавание людей и т. д.).
- opencv_calib3d — калибровка камеры, поиск стерео-соответствия и элементы обработки трехмерных данных.
- opencv_legacy — устаревший код, сохраненный ради обратной совместимости.
- opencv_gpu — ускорение некоторых функций OpenCV за счет CUDA, созданного при поддержке NVidia.

Фактически, OpenCV – это набор типов данных, функций и классов для обработки изображений алгоритмами компьютерного зрения. Основные модули библиотеки(рис.8):

cxcore - ядро. Содержит базовые структуры данных и алгоритмы:
- базовые операции над многомерными числовыми массивами;
- матричная алгебра, математические функции, генераторы случайных чисел;
- запись/восстановление структур данных в/из XML;
- базовые функции 2D графики.

CV - модуль обработки изображений и компьютерного зрения.
- базовые операции над изображениями (фильтрация, геометрические преобразования, преобразование цветовых пространств и т. д.);
- анализ изображений (выбор отличительных признаков, морфология, поиск

контуров, гистограммы);
- анализ движения, слежение за объектами;
- обнаружение объектов, в частности лиц;
- калибровка камер, элементы восстановления пространственной структуры.

Highgui - модуль для ввода/вывода изображений и видео, создания пользовательского интерфейса
- захват видео с камер и из видео файлов, чтение/запись статических изображений;
- функции для организации простого UI (все демо-приложения используют HighGUI).

Cvaux - экспериментальные и устаревшие функции
- пространственное зрение: стерео-калибрация, само-калибрация;
- поиск стерео-соответствия, клики в графах;
- нахождение и описание черт лица.

CvCam - захват видео. Позволяет осуществлять захват видео с цифровых видео-камер (поддержка прекращена и в последних версиях этот модуль отсутствует).

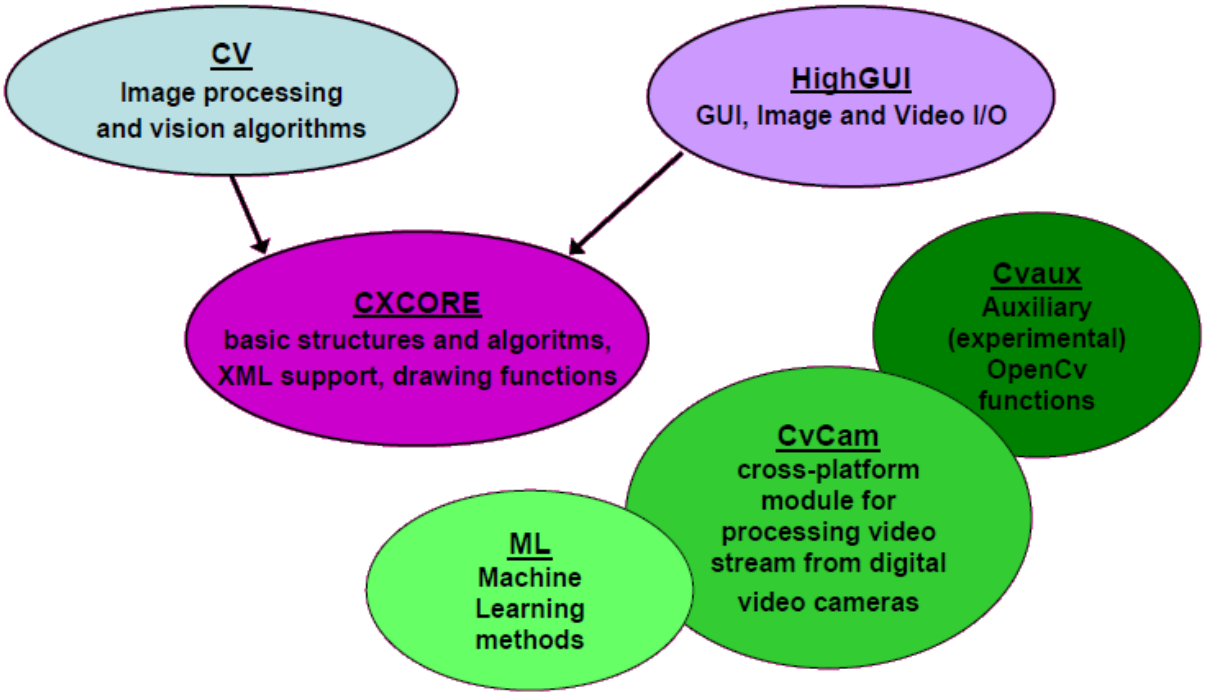

Рисунок 8 – Общий вид структуры OpenCV

2.2 Основные аспекты выбора оборудования для построения распределённой вычислительной системы

Выделение базовых требований к разрабатываемой системе является самой важной частью исследования. На данном этапе моделируется износоустойчивость модулей и компонентов, разрабатывается программное обеспечение, организующее корректную работу системы в целом при минимальном использовании памяти устройства. Как следствие, при должном выполнении указанных требований достигается максимальная производительность всех компонентов.

Подбор модулей основывается на выделении основных критериев и требований:

1. Размер устанавливаемой платы должен быть как можно меньше, в силу ограниченности пространства;
2. Оперативная память основного модуля: 512 Мб и более. Для обработки поступающего потока данных желательно ограничить выбираемый модуль указанным минимумом;
3. Возможность взаимодействия через интерфейсы GPIO или UART;
4. Наличие интерфейса USB;

В качестве основного модуля был выбран одноплатный компьютер Raspberry Pi Model B(рис.9, 10). Raspberry Pi — полноценный персональный компьютер, который оснащён всем необходимым: CPU, GPU, 256 МБ памяти, слот для флэш-карты, разъёмы для подключения клавиатуры, аудиоколонок, телевизора или монитора и других периферийных устройств. Он работает под управлением Linux и отлично производит HD-видео высокого качества или, например, позволяет играть в видеоигры.

Разработчиками Raspberry Pi являются двое английских инженеров, которые хотели сделать некое подобие мини-компьютера BBC Micro из их детства, чтобы объяснить концепцию персонального компьютера каждому ребёнку, и чтобы каждый школьник мог позволить купить себе такую игрушку для экспериментов. Компания некоммерческая, и они продают компьютеры по себестоимости.

Официальные продажи компьютера начались 29 февраля 2012 года. Спустя почти два года у Raspberry Pi появились несколько наследников, которые используют тот же принцип — плата минимального размера по минимальной цене для полноценного Linux-компьютера — но именно Raspberry Pi был первопроходцем и именно он открыл новый сегмент на рынке компьютерной техники.

На сегодняшний день выпускается в трех версиях: «A» (256 Мб ОЗУ), стоимостью $ 25, «B» ($ 35, с ethernet, 512 Мб ОЗУ) и «B+» (с четырьмя портами USB, 512 Мб ОЗУ). Разрабатывается Raspberry Pi Foundation.

Характеристики выбранного оборудования:

- Процессор: 700MHz ARMv6 (Возможен разгон до 1000 МГц).
- Память: 512MB SDRAM
- OpenGL ES 2.0
- Видео: 1080p30
- Аудио: H.264 high-profile decode
- Композитный и HDMI Видеовыход
- USB 2.0
- Слот для карты памяти SD/MMC/SDIO
- Системы ввода-вывода общего назначения (порт GPIO, пригодный для управления внешними устройствами).
- Дополнительный встроенный USB-концентратор (2 порта).
- Ethernet адаптер 10/100 Мбит (необходим для работы с интернетом).
- Open software: ОС Raspbian (Debian, скомпилированный под архитектуру ARM), веб-браузер Midori, предустановленный интерпретатор Python

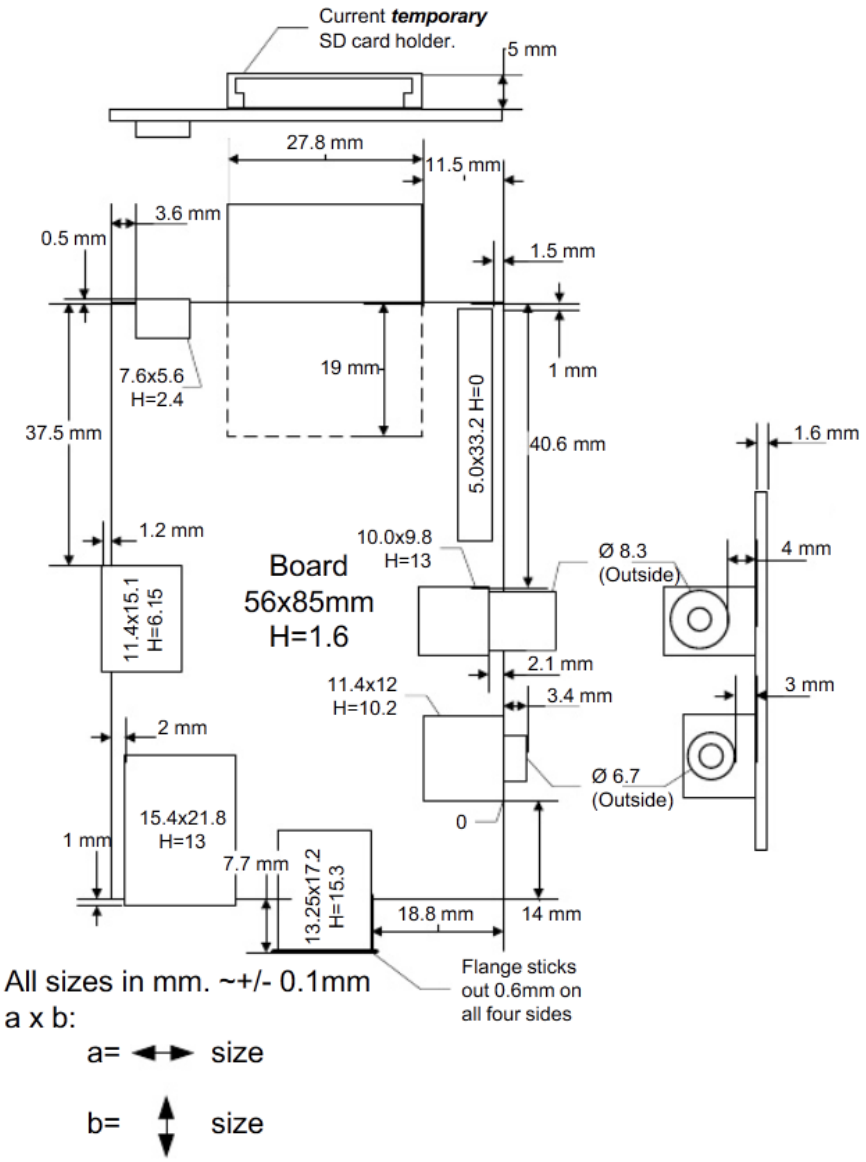

Рисунок 9 – Размеры Raspberry Pi Model B(в мм)

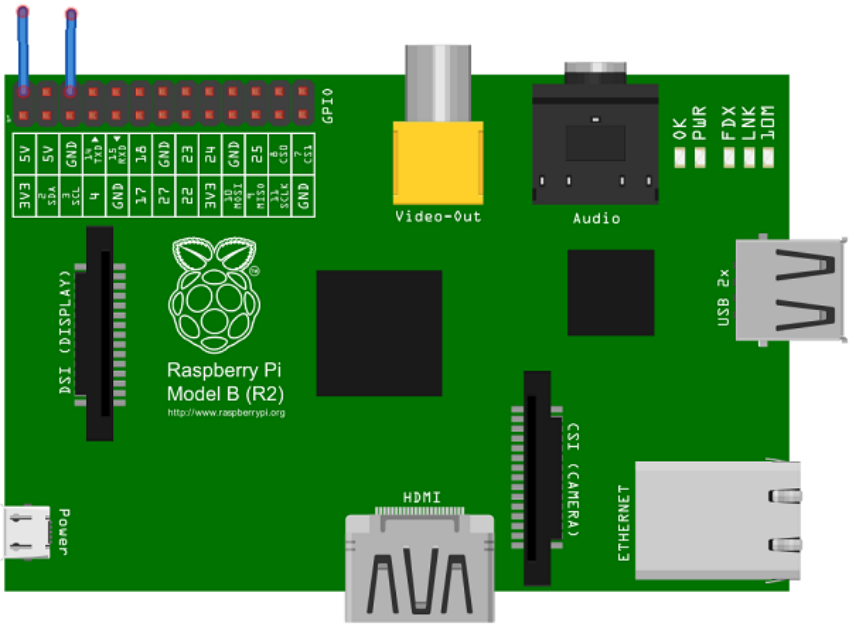

Рисунок 10 – Общий вид платы

В качестве основного вспомогательного модуля был выбран Red Back Spider Robot Controller(схема на рис.11), который разработан специально для использования большого количества сервоприводов, например, в сложных роботизированных системах. Со встроенным переключателем питания до 3А контроллер может поддерживать работу и контролировать до 48 миниатюрных сервомоторов.

Особенности:

- ATmega1280 MCU с 128K Flash, 8K SRAM и 4 КБ EEPROM
- 3A, 5V с возможностью переключения. Входное напряжение: от 7V до 30V
- 70 пинов(коннекторов) ввода-вывода, совместимые с трёхконтактными разъёмами сервоприводов(рис.11).
- Интерфейс USB и сокет ISP
- Выключатель питания и кнопка сброса
- Расстояние между пинами позволяет использовать дополнительные платы, как разработанные самостоятельно, так и поставляемые сторонними фирмами
- Поставляется со стандартным и наиболее распространённым загрузчиком Arduino
- Имеет 16 аналоговых входов по 10 бит
- Имеет до 15 ШИМ выходов (зависит от количества используемых сервоприводов)
- 4 последовательных порта (один используется интерфейсом USB)
- Возможность управления до 48 сервоприводов, при использовании стандартных библиотек Arduino

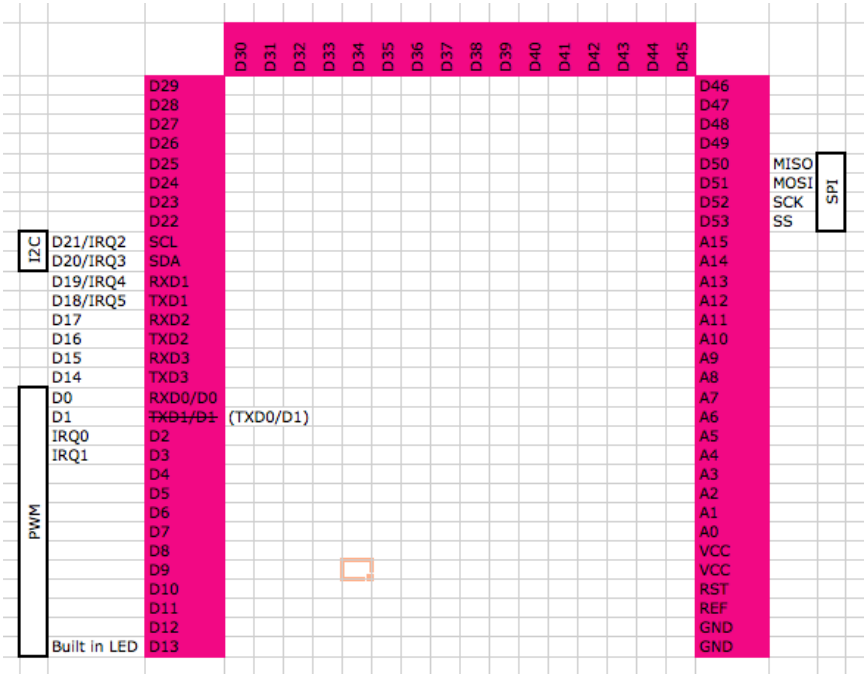

Рисунок 11 – Расположение пинов на плате Red Back Spider Robot Controller

Микроконтроллеры обычно не могут выдавать произвольное напряжение. Они могут выдать либо напряжение питания (например, 5 В), либо землю (т.е. 0 В). Но уровнем напряжения управляется многое: например, яркость светодиода или скорость вращения мотора. Для симуляции неполного напряжения используется ШИМ (Широтно-Импульсная Модуляция, англ. Pulse Width Modulation или просто PWM).

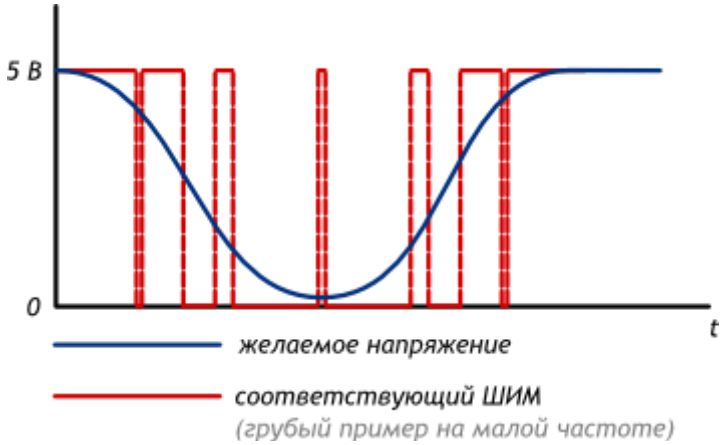

Рисунок 12 – Пример работы ШИМ

Выход микроконтроллера переключается между землёй и Vcc тысячи раз в секунду, то есть имеет частоту в тысячи герц. Человеческий глаз не замечает мерцания более 50 Гц, поэтому нам кажется, что светодиод не мерцает, а горит в полсилы.

Аналогично, разогнанный мотор не может остановить вал за миллисекунды, поэтому ШИМ-сигнал заставит вращаться его в неполную силу.

Отношение времени включения и выключения называют скважностью (англ. duty cycle). При рассмотрении сценариев при напряжении питания Vcc равным 5 вольтам, можно получить следующие графики:

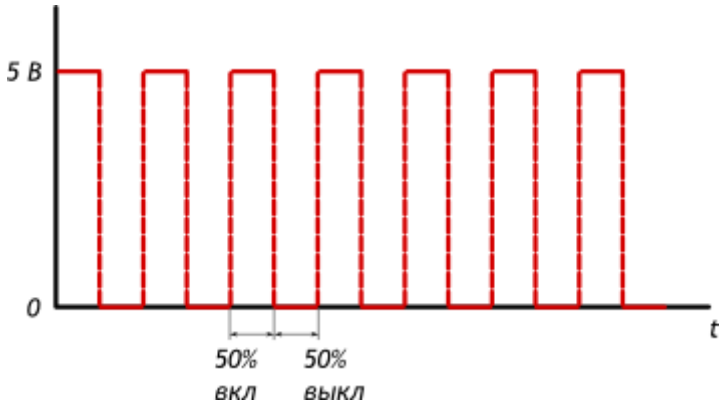

Рисунок 13 – Скважность при 2,5 В

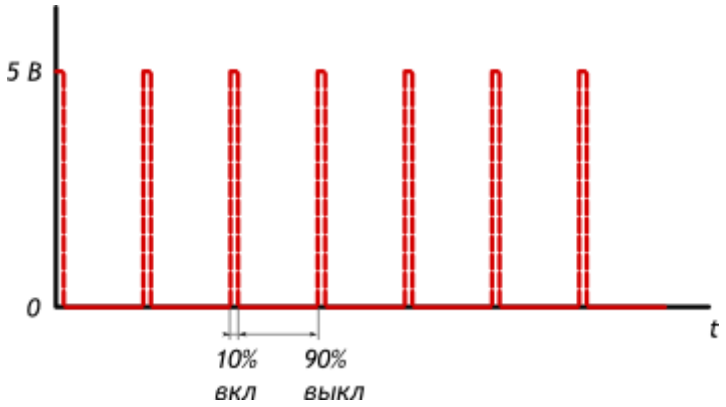

Рисунок 14 - Скважность при 0,5 В

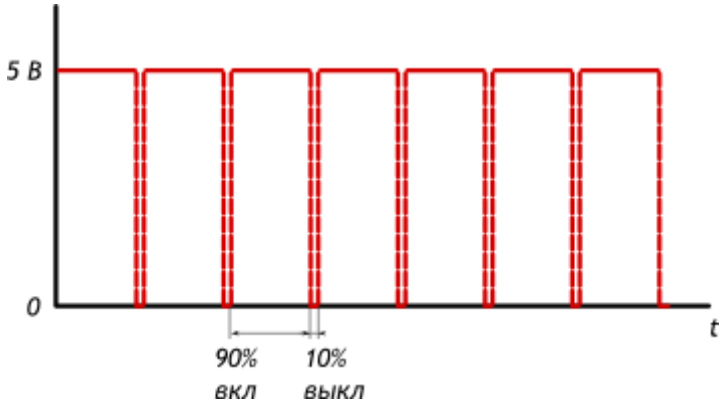

Рисунок 15 - Скважность при 4,5 В

В качестве устанавливаемых сервоприводов были выбраны моторы фирмы TowerPro MG90(рис.16). Они относятся к классу микро-серво и имеют следующие спецификации:

- Размер: 23.2 × 12.5 × 22 мм
- Вес:14 г
- Скорость поворота: 0.12 сек/60 градусов (при питании 4.8В) и 0.10сек/60градусов (при питании 6В)
- Рабочее напряжение: от 4.8В до 6В
- Аналоговое подключение
- Максимальный угол поворота: 120 градусов
- Материал вала: металл
- Материал шестерёнок: металл
- Длина подключаемого провода: 20 см

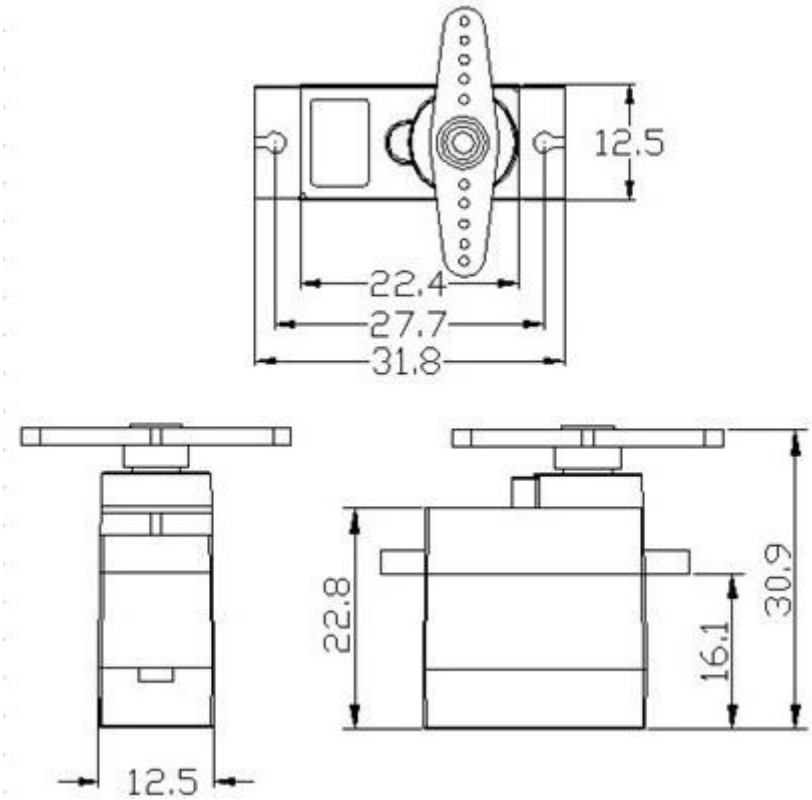

Рисунок 16 – Размер сервоприводов

Основным плюсом выбора являются металлические шестерни, которые имеют повышенную износоустойчивость, а так же наименьший люфт по сравнению с пластмассовыми.

Выводы по второй главе

При разработке системы носящий аппаратный характер необходимо учитывать возможности как каждого модуля по отдельности, так и при работе всей конструкции в целом. Любой модуль подбирается с учётом возможностей предыдущего: они должны либо дополнять друг друга, либо заменять (при необходимости). Функциональность программного обеспечения не менее важна – при должных настройках даже самые простые в использовании фреймворки обеспечивают колоссальную производительность системы в целом, обеспечивая её связность и гибкость.

3 РАЗРАБОТКА И ОБОСНОВАНИЕ ПРОЕКТНО-ПРАКТИЧЕСКИХ РЕКОМЕНДАЦИЙ ПО РЕАЛИЗАЦИИ БАЗОВЫХ ВОЗМОЖНОСТЕЙ РАСПРЕДЕЛЁННОЙ ВЫЧИСЛИТЕЛЬНОЙ СИСТЕМЫ

3.1 Разработка проектных рекомендаций по реализации базовых возможностей распределённой вычислительной системы с целью выделения основных требований к распределению нагрузки

Одним из самых важных этапов разработки сложной распределённой вычислительной системы с аппаратной реализацией является правильное построение каркаса(рис. 17-19). Однако это же является и одним из самых творческих моментов разработки. В качестве элемента исследования была выбрана сложная конструкция, требующая наличия алгоритмов оперирования шестью «конечностями» - гексапод(«hex» - греч., шесть)

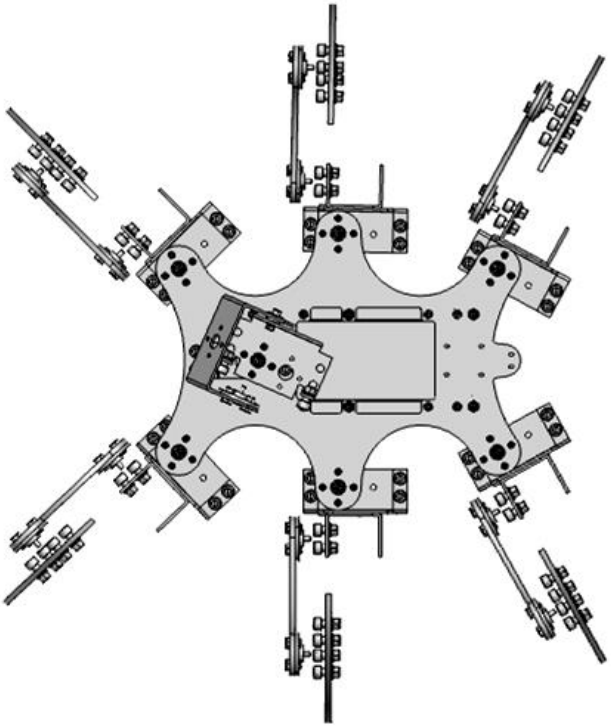

Рисунок 17 – Вид каркаса сверху

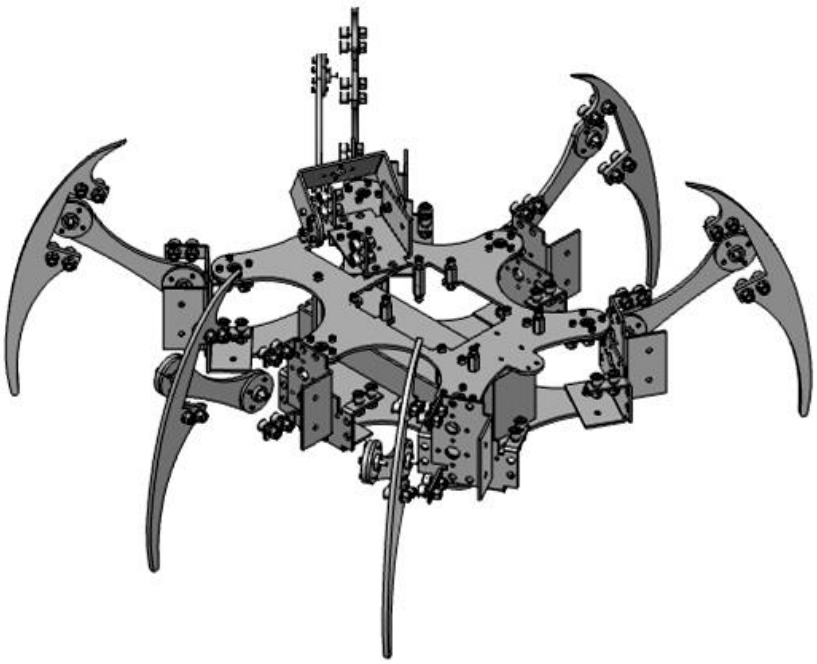

Рисунок 18 – Изометрический вид каркаса

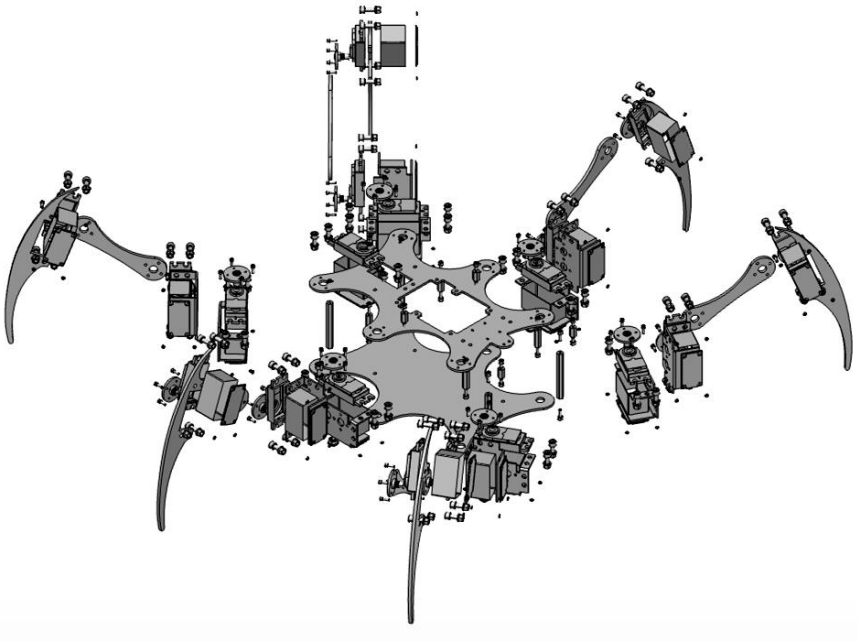

Рисунок 19 – Детализированная развёртка каркаса

Разработка и моделирование планируемых нагрузок для подобных конструкций обычно проводят в САПР. Система автоматизированного проектирования (САПР) - автоматизированная система, реализующая информационную технологию выполнения функций проектирования, представляет собой организационно-техническую систему, предназначенную для автоматизации процесса проектирования, состоящую

из персонала и комплекса технических, программных и других средств автоматизации его деятельности.

Самым распространённым примером является SolidWorks (Солидворкс) — программный комплекс САПР для автоматизации работ промышленного предприятия на этапах конструкторской и технологической подготовки производства. Обеспечивает разработку изделий любой степени сложности и назначения.

Размеры всех конструкций, как правило, следует подбирать не только с учётом расположения всех плат, но и не исключая возможности расширения системы (подключения новых модулей). После определения каркаса и комплектации наступает этап разработки. Так как общая конструкция имеет нестандартное строение, то все исследования и наработки будут основываться на строении ног насекомого(рис.20):

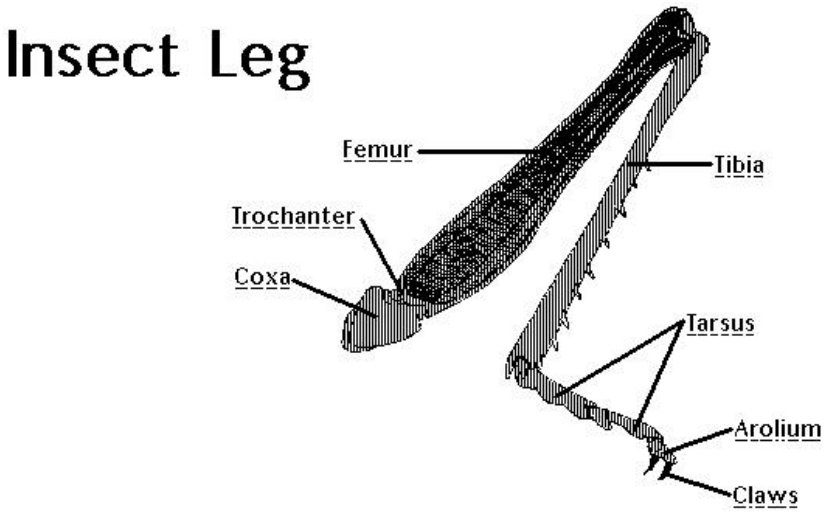

Рисунок 20 – Строение ноги насекомого

Данное решение обосновано одним из основных принципов программирования: при написании кода программист должен учесть возможность его доработки кем-либо другим, а следовательно, код должен быть лёгок к восприятию. Исходя из вышеуказанного, соответствующим сервоприводам были даны следующие метки: Coxa, Femur и Tibia.

Первым важным элементом изучения и построения системы будет разработка модели движения ног с последствующей реализацией.

Пусть в пространстве (x, y, z) имеется некая точка A, к которой следует дотянуться одной из «ног». Тогда общий вид конструкции сверху обретает следующий вид(рис.21):

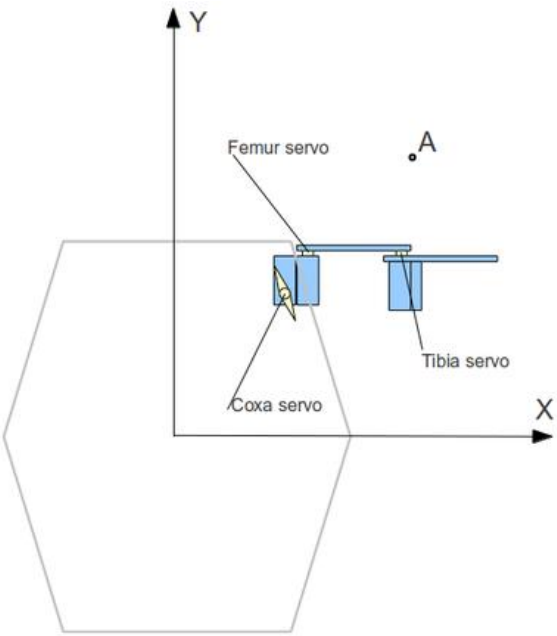

Рисунок 21 – Точка А в пространстве

Следовательно, для достижения данной точки требуется повернуть только один сервопривод. То есть итог должен быть следующим:

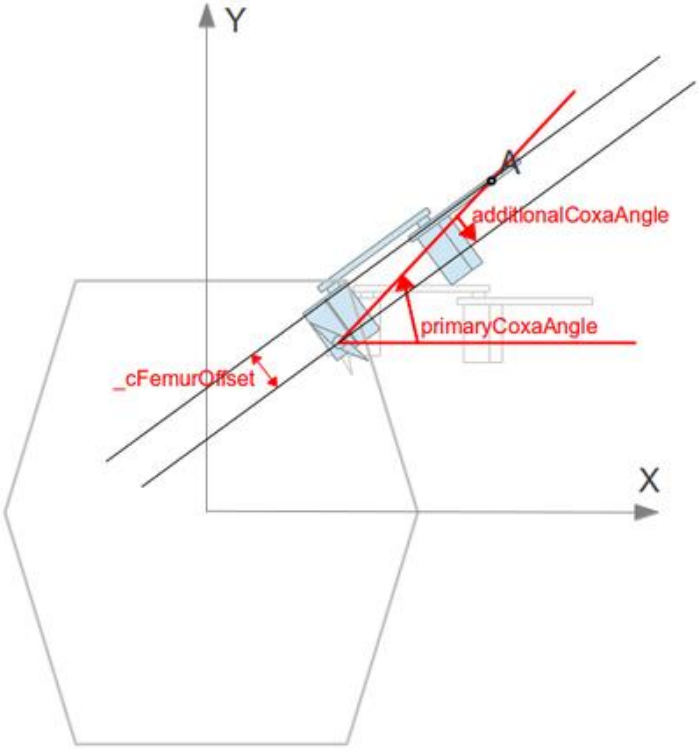

Рисунок 22 – Достижение точки А

Здесь видно, что центр поворота Coxa не находится на одной линии с центрами Femur и Tibia из-за расстояния между осью Coxa и осью Femur - *_cFemurOffset*.

На рисунке 21 изображён *primaryCoxaAngle* — угол (0; A) к оси X (что эквивалентно углу точки A в полярных координатах). Но на схеме видно, что при этом сама нога — не расположена под этим углом. Для достижения конечной точки сначала необходимо найти расстояние до неё по следующей формуле:

$$D = \sqrt{(x - Sx)^2 + (y - Sy)^2} \qquad (1),$$

где x – расположение конечной точки относительно оси X, Sx – координаты точки крепления (и центра вращения) Coxa. Соответственно, y и Sy - конечной точки относительно оси Y и координаты точки крепления.

Далее производится расчет дополнительного угла поворота:

additionalCoxaAngle = _cFemurOffset/D (2)

Следующим шагом определяется полярный угол по координатам (x - Sx) и (y - Sy).

Каждая точка в полярной системе координат может быть определена двумя полярными координатами, что обычно называются r (радиальная координата, угловое расстояние, встречается вариант ρ) и φ (угловая координата, полярный угол, азимут, позиционный угол, иногда пишут θ или t). Координата r соответствует расстоянию до полюса, а координата φ равна углу в направлении против часовой стрелки от луча через 0° (иногда называется полярной осью).

Полярный радиус определен для любой точки плоскости и принимает неотрицательные значения $r \geqslant 0$. Полярный угол φопределен для любой точки плоскости, за исключением полюса O, и принимает значения $-\pi < \varphi \leqslant \pi$. Полярный угол измеряется в радианах и отсчитывается от полярной оси:

- в положительном направлении (против направления движения часовой стрелки), если значение угла положительное;
- в отрицательном направлении (по направлению движения часовой стрелки), если значение угла отрицательное. [18]

Исходя из полученных данных, конечный угол поворота будет выглядеть как:

cAngle = primaryCoxaAngle - additionalCoxaAngle - _cStartAngle (3),

где _cStartAngle – стартовый угол поворота сервопривода.

При этом задача сводится к поиску точки пересечения двух окружностей. Одна — в точке, откуда «растет» Femur, вторая — точка, куда требуется дотянуться (с локальным 2d координатами). Радиусы окружностей — длины Femur и Tibia соответственно. Если окружности пересекаются — то

в одной из 2х точек можно расположить сустав. Как правило, всегда выбирается верхняя, чтобы «колени» у гексапода были выгнуты вверх, а не вниз. Если не пересекаются — то целевая точка достигнута не будет.

Значит, исходя из задачи о пересечении двух окружностей, необходимо найти все их точки пересечения (либо одна, либо две, либо ни одной точки, либо окружности совпадают).

Предположим, не теряя общности, что центр первой окружности - в начале координат (если это не так, то перенесём центр в начало координат, а при выводе ответа будем обратно прибавлять координаты центра). Тогда мы имеем систему двух уравнений:

$x2 + y2 = r12$

$(x - x2)2 + (y - y2)2 = r22$

Вычтем из второго уравнения первое, чтобы избавиться от квадратов переменных:

$x2 + y2 = r12$

$x (-2x2) + y (-2y2) + (x22 + y22 + r12 - r22) = 0$

Таким образом, мы свели задачу о пересечении двух окружностей к задаче о пересечении первой окружности и следующей прямой:

$Ax + By + C = 0,$

$A = -2x2,$

$B = -2y2,$

$C = x22 + y22 + r12 - r22.$

Результатом вычислений будет траектория движения, представленная на рисунке 23.

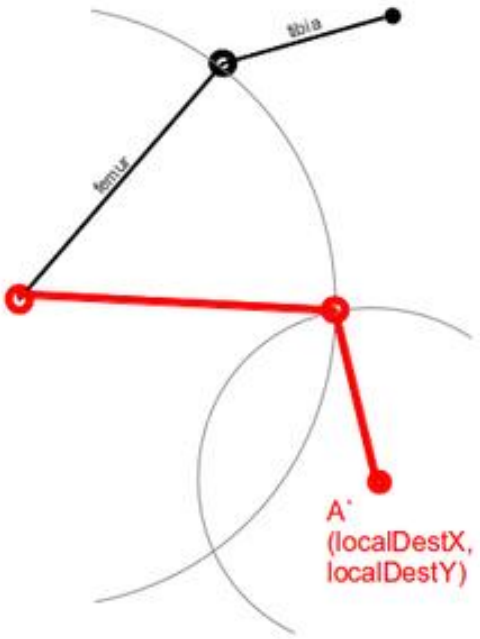

Рисунок 23 – Графическое представление движения ног вверх

3.2 Обоснование предлагаемых рекомендаций, предоставление вариантов их реализации

Raspberry Pi Model B является оптимальным решением за счёт изначального наличия многих необходимых компонентов, таких как Ethernet и рекомендуемый размер ОЗУ: 512 Мб. Для сравнения, Raspberry Pi Model A не имеет встроенного Ethernet(требуется дополнительный модуль), а так же размер ОЗУ не соответствует минимальным требованиям – в данной модели всего 256 Мб.

Для разработки 3dof гексапода нужно 3*6 = 18 сервомоторов и раздельных каналов управления ими, для чего рассматривались достаточно популярные платы Arduino Mega 2560 и их аналоги, т.к. количество ШИМ-выходов, которыми можно управлять сервоприводами , удовлетворяет описанным требованиям. Однако учитывая возможность дальнейшего развития системы, был выделен наиболее выгодный вариант среди аналогов Arduino Mega 2560 - это плата от Dagu, Red Back Spider Controller.

Она предлагает все свои выходы в виде готовых 3-х штырьков (земля, питание, сигнал), и развязку по питанию. Питание самого контроллера стабилизировано, а на разъемы сервоприводов идет как есть, то есть в размере 5 В. Так же серия небольших экспериментов показала, что питание сервоприводов развязано с питанием контроллера, т.к. помех в работу контроллера 18 одновременно работающих сервомоторов не вносят. Это позволяет просто подать на клемму питания 7-30 В достаточной мощности, что избавляет от проблемы реализации раздельного питания логики и моторов.

Так как Raspberry Pi Model B располагает наличием пинов с различным питанием(3,3 В и 5 В соответственно)[20], это позволяет подключить Red Back Spider Controller напрямую, без использования макетной платы и дополнительных элементов стабилизации питания.

На рисунке 24 представлен общий вид пинов Raspberry Pi Model B с их последующим подключением. Рисунок 25 отображает пример соединения с Arduino Mega 2560.

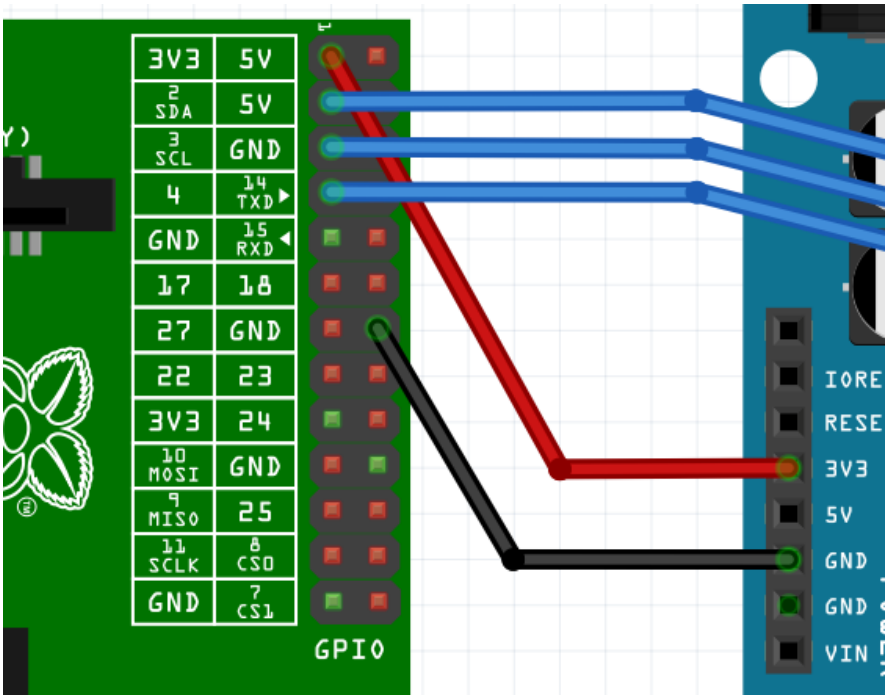

Рисунок 24 – Пины Raspberry Pi Model B

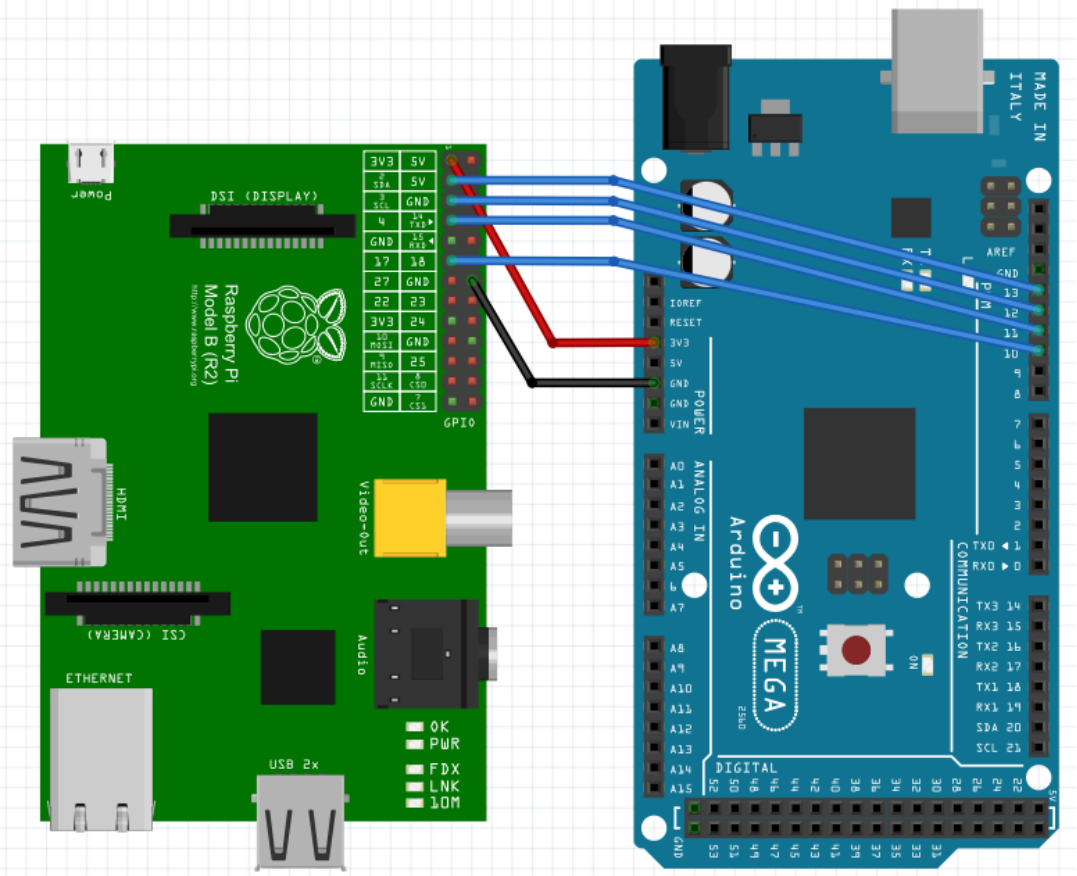

Рисунок 25 – Соединение Raspberry Pi Model B и Arduino Mega 2560

После подключения плат и процесса установки WebIOPi(описанного во второй главе), требуется написать несколько коротких скриптов, определяющих пины Raspberry Pi и запускающие необходимые процессы с нужными параметрами.

Для управления электромоторами, подключенными к плате Red Back Spider Controller, используются четыре выхода Raspberry Pi. Их назначение следующее:

```
# Left motor GPIOs
LEFT_GO_PIN = 17 #PWM сигнал скорости
LEFT_DIR_PIN = 4 #направление движения

# Right motor GPIOs
RIGHT_GO_PIN = 10  #PWM
RIGHT_DIR_PIN = 25 #направление движения
```

Web-камера работает в двух режимах – передача потокового видео с минимальным разрешением(в целях экономии ресурсов) с помощью программы mjpg-streamer и захват изображения с последующим сохранением с максимальным разрешением. Выбор режима осуществляется запуском одного из скриптов через Web-интерфейс: stream.sh, stream_stop.sh, photo.sh.

Скрипт фотографирования (photo.sh):

```
#!/bin/sh
fswebcam -d /dev/video0 -p MJPEG -r 640x480  --jpeg 95 --shadow --title "cambot" --subtitle "Front camera" --info "" --save /usr/share/webiopi/htdocs/app/Raspirobot/ph.jpg -q
```

Скрипт включения web-камеры (stream.sh):

```
#!/bin/sh
STREAMER=mjpg_streamer
DEVICE=/dev/video0
RESOLUTION=160x120       #320x240
FRAMERATE=25
HTTP_PORT=8001
# check for existing webcam device
if [ ! -e "/dev/video0" ]; then
  echo "stream.sh: Error - NO /dev/video0 device" 2>&1 | logger
  exit 2
fi
PLUGINPATH=/home/pi/mjpg-streamer-r63
```

"$PLUGINPATH/$STREAMER" -i "$PLUGINPATH/input_uvc.so -n -d $DEVICE -r $RESOLUTION -f $FRAMERATE" -o "$PLUGINPATH/output_http.so -n -p $HTTP_PORT" –b

Скрипт отключения камеры (stream_stop.sh):

```
#!/bin/sh
kill -9 `pidof mjpg_streamer`
```

Создание web-интерфейса для WebIOPi является достаточно нетрудоёмким процессом. Все макросы, написанные на Python, вызываются при помощи функций из javascript, закреплёнными на элементах управления web-интерфейса, то есть кнопками. Например:

```
button = webiopi().createButton("bt_up", "/\\", go_forward, stop);
$("#up").append(button);
```

Функция go_forward выглядит следующим образом:

```
function go_forward() {
   webiopi().callMacro("go_forward");
}
```

т.е. в данном случае, нажатие кнопки «Вперёд» вызовет макрос go_forward.

Поскольку использование компьютерной мыши не всегда удобно конечному пользователю, возможно продублировать управление с помощью стрелок на клавиатуре:

```
$(document).keydown(function(e)
{
   switch(e.which)
    {
      case 37:turn_left(); break;
      case 38:go_forward(); break;
      case 39:turn_right(); break;
      case 40:go_backward(); break;
      case 32:stop(); break;
      case 75:camera(); break;
      case 80:photo(); break;
   }
});
```

Видео и фото в интерфейсе транслируются в соответствующее окно после выполнения макроса:

```
function camera() {
 $("#vid").html('');
 webiopi().callMacro("camera");
 $("#vid").html('<img width="320" height="240"
src="http://raspberrypi.local:8001/?action=stream">');
}

function photo() {
 $("#ph").html('');
 webiopi().callMacro("photo");
 $("#ph").html('<img src="Raspirobot/ph.jpg">');
}
```

После написания необходимых скриптов можно приступить к конфигурированию WebIOPi. [20]

Файл конфигурации находится в папке /etc/webiopi/. Он состоит из нескольких блоков. Прежде всего интерес представляет блок, описывающий макросы:

```
[SCRIPTS]
# Load custom scripts syntax :
# name = sourcefile
#   each sourcefile may have setup, loop and destroy functions and macros
cambot =/usr/share/webiopi/htdocs/app/Raspirobot/cambot.py
```

В этот блок надо добавить строку в формате name=sourcefile со скриптом Python, содержащим макросы.

И второй блок, представляющий интерес – конфигурация сервера. В нём задаётся номер порта, на котором будет работать сервер, путь к файлу с паролем и корневую директорию для сервера.

```
[HTTP]
# HTTP Server configuration
enabled = true
port = 8000

# File containing sha256(base64("user:password"))
# Use webiopi-passwd command to generate it
passwd-file = /etc/webiopi/passwd

# Use doc-root to change default HTML and resource files location
#doc-root = /home/pi/webiopi/examples/scripts/macros
```

```
# Use welcome-file to change the default "Welcome" file
#welcome-file = index.html
```

После несложного процесса проведения конфигурации, последним этапом является написание короткого скрипта на Python.

Для WebIOPi скрипт должен содержать функции setup и destroy, которые будут вызываться при запуске сервера и его выключении соответственно. Как правило, в этих функциях определяется режим работы и состояние портов ввода-вывода.

Функции, которые должны быть доступны в web интерфейсе, должны предваряться идентификатором @webiopi.macro. Например, так выглядит описание функции go_forward, вызывающей движение гексапода вперед:

```
@webiopi.macro
def go_forward():
left_forward()
right_forward()
```

Для управления web-камерой используется команда call, вызывающая соответствующий скрипт:

```
def camera_start():
return_code = call("/usr/share/webiopi/htdocs/app/Raspirobot/stream.sh", shell=True)
```

Выводы по третьей главе

Для корректной работы сложной распределённой вычислительной системы необходим не только тщательный подбор оборудования и программного обеспечения, но и должная настройка для реализации оркестрации сервисов. Однако перед тем как приступить данному этапу, необходимо провести моделирование работы системы в соответствующих программных комплексах. Наиболее удобным вариантом в этом случае является Fritzing – бесплатное ПО с возможностью добавления собственных плат и программного кода.

ЗАКЛЮЧЕНИЕ

Изучение и последующее построение собственной сложной распределённой вычислительной системы является очень увлекательным процессом. Учитывая актуальность данного направления в наши дни и высокую востребованность на рынке можно сказать, что подобные технологии будут активно развиваться и дальше, причём в самых разных областях: от разработки и поддержки программного обеспечения для крупных компаний, до создания сложнейших роботизированных систем.

Базируясь на простых концепциях SOA и принципах SDP становится возможным реализация многих сложных проектов. Правила распределённых вычислительных систем обеспечат высокую производительность при низком потреблении ресурсов, организуя равномерную нагрузку на все элементы системы, а следовательно, минимизируя затраты компаний на закуп оборудования.

В данной диссертационной работе было проведено исследование, посвящённое возможности объединения принципов SOA и SDP на аппаратном уровне, в результате чего были выявлены преимущества обеих сторон:

Основываясь на концепциях SOA, были разработаны и успешно внедрены многие элементы системы, а так же были реализованы возможности непрерывного улучшения как аппаратной так и программной части. Были использованы текущие разработки сторонних компаний, предоставленные в сети Интернет в свободном доступе и имеющие открытый исходный код, что так же является одним из тактических преимуществ SOA.

Основываясь на принципах SDP, была создана «сервисная прослойка», организующая режим горячего резервирования системы, то есть в случае выхода из строя одного из модулей, это не повлияет на производительность в целом, и позволит проработать в штатном режиме до проведения технического обслуживания.

Система основана на специально адаптированном дистрибутиве Linux - Raspbian Wheezy, что предоставляет ей дополнительную гибкость и возможности расширения для доработок в будущем. При необходимости, данная операционная система способна мигрировать на ROS (Robotic Operation System) версии Hydro, которая имеет ряд преимуществ для дальнейшего развития сложной распределённой вычислительной системы в виде более детального контроля аппаратной части, в том числе на низких уровнях. Это является доказательством реализации общего принципа SOA и SDP – возможность обновления и совершенствования системы, повышение управляемости и показатель масштабируемости.

По итогам исследования был сделан заключительный вывод: принципы SDP в большинстве своём предполагают реализацию сервисов и услуг, которые будут предоставляться компаниями на рынке конечным пользователям и абонентам. Концепция SOA является основой для

построения и реализации возможностей SDP, однако может использоваться как отдельная архитектура в других целях.

Объединение основных преимуществ данных архитектур и вынесение их на отдельный аппаратный уровень является реальным, что предоставляет отличную платформу для разработки и исследования новых возможностей сложных распределённых вычислительных систем в будущем.

СПИСОК ИСПОЛЬЗОВАННОЙ ЛИТЕРАТУРЫ

1. Jason M. O'Kane «A Gentle Introduction to ROS» - ISBN 978-14-92143-23-9 ©2014, Jason Matthew O'Kane
2. Mell, Peter and Grance, Timothy. The NIST Definition of Cloud Computing (англ.). Recommendations of the National Institute of Standards and Technology. NIST (20 October 2011)
3. Martin, Richard J. and Hoover, Nicholas. Guide To Cloud Computing
4. Кабатов Д.А. «Что такое облачные сервисы для небольших компаний?» URL: http://www.moysklad.ru/chto_takoe_oblachnye_servisy/
5. Облачные технологии http://ru.wikipedia.org/
6. Mark Carlson (Oracle), Martin Chapman (Oracle), Alex Heneveld (Cloudsoft), Scott Hinkelman (Oracle), Duncan Johnston-Watt (Cloudsoft), Anish Karmarkar (Oracle), Tobias Kunze (Red Hat), Ashok Malhotra (Oracle), Jeff Mischkinsky (Oracle), Adrian Otto (Rackspace), Vivek Pandey (CloudBees), Gilbert Pilz (Oracle), Zhexuan Song (Huawei), Prasad Yendluri (Software AG). Cloud Application Management for Platforms. Version 1.0 URL: http://www.cloudspecs.org/CAMP/CAMP_v1-0.pdf
7. Gillam, Lee. Cloud Computing: Principles, Systems and Applications / Nick Antonopoulos, Lee Gillam. — L.: Springer, 2010. — 379 p. — (Computer Communications and Networks). — ISBN 9781849962407
8. OASIS Reference Model for Service Oriented Architecture V 1.0 URL: http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf
9. Клив Финкельштейн (Clive Finkelstein) "Корпорация: сервис-ориентированная архитектура" (The Enterprise: Service-Oriented Architecture (SOA). URL: http://www.dmreview.com/article_sub.cfm?articleID=1016488
10. Джерими Уэстерман (Jeremy Westerman) "Сервис-ориентированная архитектура сегодня: введение в SOA" (SOA Today: Introduction to Service-Oriented Architecture). URL: http://www.dmreview.com/article_sub.cfm?articleID=1016488
11. Джерими Уэстерман (Jeremy Westerman) "Сервис-ориентированная архитектура сегодня: значение SOA для бизнеса" (SOA Today: Business Value of SOA). URL: http://www.dmreview.com/article_sub.cfm?articleID=7992
12. Материалы, опубликованные на сайте Консорциума по интеграции (Integration Consortium). URL:http://www.integrationconsortium.org/
13. Yefim V. Natis, Massimo Pezzini « Twelve Common SOA Mistakes and How to Avoid Them » URL:http://www.gartner.com/it/content/754400/754413/twelve_common_soa_mistakes.pdf
14. Материалы, опубликованные в документации ООО «Фирма «СВЕТЕЦ»» «Подсистема управления и исполнения real-time процессов. Техническое описание» 2013 г.

15. Eric Trouch - https://code.google.com/p/webiopi/
16. openFrameworks http://openframeworks.cc/
17. Case Studies Bundle - Practical Python and OpenCV
18. Полярная система координат https://ru.wikipedia.org/
19. https://inductible.wordpress.com/2013/02/23/telemus-a-remotely-operated-vehicle-based-on-raspberry-pi-and-arduino/
20. Конфигурирование WebIOPi http://habrahabr.ru/post/181930/

ПРИЛОЖЕНИЕ А
Основной файл

```
#include <Servo.h>
#include "Leg.h"

void log(const char* x) { Serial.println(x); }
void log(float x) { Serial.println(x); }
void log(int x) { Serial.println(x); }

static Leg legs[6];
static int N = 6;

static Leg* rightLegs = legs;
static Leg* leftLegs = legs + N / 2;

static Point zero(0,0,0);

static Point stateLinearMovement;

Servo motor;
Servo myservo;
int pwm_a = 3;
int pwm_b = 11;
int dir_a = 12;
int dir_b = 13;

int pwm_a1 = 4;
int pwm_b1 = 5;
int dir_a1 = 6;
int dir_b1 = 7;

int val1=0;
int val2=0;
int val3=0;
int val4=0;
int pos=0;

void processState()
{
   for (int i = 0; i < N; ++i)
      legs[i].reachRelativeToCurrent(stateLinearMovement);
}
```

```
void walk(int steps, Point direction)
{
   for (int i = 0; i < steps; i++)
   {
      for(float p = 0; p <= 2; p += 0.025)
      {
         float progress;
         float height1;
         float height2;
         if (p < 1)
         {
            progress = p;
            height1 = 0;
            height2 = direction.z * (0.5 - fabs(0.5 - p));
         }
         else
         {
            progress = 1 - (p - 1);
            height1 = direction.z * (0.5 - fabs(1.5 - p));
            height2 = 0;
         }

         Point group1(- direction.x / 2 + (direction.x - progress * direction.x),
                  - direction.y / 2 + (direction.y - progress * direction.y),
                  height1);
         Point group2(- direction.x / 2 + progress * direction.x,
                  - direction.y / 2 + progress * direction.y,
                  height2);

         if (legs[0].getCurrentRelative().maxDistance(group1) > 10)
            smoothTo(group1, 0);
         if (legs[1].getCurrentRelative().maxDistance(group2) > 5)
            smoothTo(group2, 1);

         for (int li = 0; li < N; li+=2)
         {

            legs[li].reachRelativeToDefault(group1);
            legs[li + 1].reachRelativeToDefault(group2);
         }

         delay(1);
      }
   }
```

```
}

void smoothTo(Point& to)
{
   smoothTo(to, 0);
   smoothTo(to, 1);
}

void smoothTo(Point& to, int legGroup)
{
   Point relative[N];
   for (int i = legGroup; i < N; i += 2)
      relative[i].assign((legs[i].getDefaultPos()          +          to)          -
legs[i].getCurrentPos());

   for(float p = 0; p <= 1; p += 0.05)
   {
      for (int li = legGroup; li < N; li += 2)
      {
         Point currSubStep = relative[li] * p;
         Point currStep = legs[li].getCurrentPos() + currSubStep;
         currStep.z = legs[li].getDefaultPos().z + to.z + 50 * (0.5 - fabs(0.5 -
p));
         legs[li].reach(currStep);
      }

      delay(5);
   }
}

void configureLegs()
{
   rightLegs[0].attach(46, 47, 39);
   rightLegs[1].attach(40, 41, 42);
   rightLegs[2].attach(43, 44, 45);

   leftLegs[0].attach(29, 28, 36);
   leftLegs[1].attach(35, 34, 33);
   leftLegs[2].attach(32, 31, 30);


   rightLegs[0].configureServoDirections(-1, -1,  1, false);
   leftLegs [0].configureServoDirections(-1,  1, -1, false);
```

```
rightLegs[1].configureServoDirections(-1, -1,  1, false);
leftLegs [1].configureServoDirections(-1,  1, -1, true);

rightLegs[2].configureServoDirections(-1, -1,  1, false);
leftLegs [2].configureServoDirections(-1,  1, -1, true);

for (int i = 0; i < 3; i++)
{
   rightLegs[i].configureFemur(-26, 12, 60, deg2rad(10));
   leftLegs [i].configureFemur(-26, 12, 60, deg2rad(10));

   rightLegs[i].configureTibia(58, deg2rad(-70));
   leftLegs [i].configureTibia(58, deg2rad(-70));
}

rightLegs[0].configureCoxa( 34,  65, deg2rad(      20),  10);
leftLegs [0].configureCoxa(-34,  65, deg2rad(180 - 20), -10);

rightLegs[1].configureCoxa( 52,   0, deg2rad(-         20),  10);
leftLegs [1].configureCoxa(-52,   0, deg2rad(- (180 - 20)), -10);

rightLegs[2].configureCoxa( 34, -65, deg2rad(-         63),  10);
leftLegs [2].configureCoxa(-34, -65, deg2rad(- (180 - 63)), -10);

rightLegs[0].configureDefault(Point( 94, 120, -70), true);
leftLegs [0].configureDefault(Point(-94, 120, -70), true);

rightLegs[1].configureDefault(Point( 120, 00, -70), true);
leftLegs [1].configureDefault(Point(-120, 00, -70), true);

rightLegs[2].configureDefault(Point( 94, -110, -70), true);
leftLegs [2].configureDefault(Point(-94, -110, -70), true);

rightLegs[0].tuneRestAngles(PI / 2,
                            PI / 2 + deg2rad(10),
                            PI / 2);
leftLegs[0].tuneRestAngles(PI / 2,
                            PI / 2 - deg2rad(5),
                            PI / 2);

leftLegs[1].tuneRestAngles(PI / 2,
                            PI / 2,
                            PI / 2 - deg2rad(10));
```

```
    rightLegs[1].tuneRestAngles(PI / 2,
                                PI / 2,
                                PI / 2 + deg2rad(20));


    leftLegs[2].tuneRestAngles(PI / 2,
                                PI / 2 - deg2rad(10),
                                PI / 2);

    for (Leg* leg = legs; leg < legs + 6; leg++)
        leg->reachRelativeToDefault(zero);
}

void setup()
{
    Serial.begin(9600);

    Serial.println("Starting");

    configureLegs();

 pinMode(pwm_a1, INPUT);
  pinMode(pwm_b1, INPUT);
  pinMode(dir_a1, INPUT);
  pinMode(dir_b1, INPUT);

  pinMode(pwm_a, OUTPUT);
  pinMode(pwm_b, OUTPUT);
  pinMode(dir_a, OUTPUT);
  pinMode(dir_b, OUTPUT);


  for (int i = 0; i < 6; i++)
   legs[i].detach();

}


void loop()
{
 val1=digitalRead(pwm_a1);
   val2=digitalRead(pwm_b1);
   val3=digitalRead(dir_a1);
   val4=digitalRead(dir_b1);
```

```
    if(val1 == 1&val2==1&val3==1&val4==1)
    {
      digitalWrite(dir_a, 1);
      digitalWrite(dir_b, 1);
      walk(2, Point(0, -80, 50));
      walk(2, Point(70, 0, 50));
      smoothTo(zero);

       };

    if(val1 == 1&val2==0&val3==1&val4==0)
    {
     digitalWrite(dir_a, 0);
     digitalWrite(dir_b, 0);
     analogWrite(pwm_a, 255);
     analogWrite(pwm_b, 255);
    };

    if(val1 == 1&val2==1&val3==1&val4==0)
    {
     analogWrite(pwm_a, 0);
     analogWrite(pwm_b, 255);
    };

    if(val1 == 1&val2==0&val3==1&val4==1)
    {
     analogWrite(pwm_a, 255);
     analogWrite(pwm_b, 0);
    };

    if(val1 == 0&val2==0&val3==0&val4==0)
    {
    analogWrite(pwm_a, 0);
    analogWrite(pwm_b, 0);
    };
}
```

ПРИЛОЖЕНИЕ Б
Файл Leg.h

```
#ifndef LEG_H__
#define LEG_H__

#include <math.h>
#include <Servo.h>

#define sqr(x) ((x)*(x))
#define PI 3.141592653589793238
#define rad2deg(x) ( (180.0 / PI) * (x) )
#define deg2rad(x) ( (PI / 180.0) * (x) )
#define fabs(x) ((x) >= 0 ? (x) : - (x))
#define max(x, y) ((x) > (y) ? (x) : (y))

#define DONT_MOVE 123456.123456

void log(const char* x);
void log(const char* x, int a);
void log(const char* x, float a);
void log(const char* x, float a, float b);
void log(const char* x, float a, float b, float c);
void log(int x);
void log(float x);

float polarAngle(float x, float y, bool thirdQuarterFix)
{
   if (x > 0)
      return atan(y / x);

   if (x < 0 && y >= 0)
   {
      if (thirdQuarterFix)
         return atan(y / x) - PI;
      return atan(y / x) + PI;
   }

   if (x < 0 && y < 0)
      return atan(y / x) - PI;

   if (y > 0)
      return PI / 2;
```

```
    if (y < 0)
        return PI / -2;

    return 0;
}

struct Point
{
    Point()
    {}

    Point(float _x, float _y, float _z)
    {
        x = _x;
        y = _y;
        z = _z;
    }

    float x, y, z;

    Point operator+(Point& that)
    {
        return Point(x + that.x,
                     y + that.y,
                     z + that.z);
    }

    Point operator-(Point& that)
    {
        return Point(x - that.x,
                     y - that.y,
                     z - that.z);
    }

    Point operator*(float m)
    {
        return Point(x * m,
                     y * m,
                     z * m);
    }

    void operator=(Point& that)
    {
        x = that.x;
```

```
        y = that.y;
        z = that.z;
    }

        void assign(Point that)
    {
        x = that.x;
        y = that.y;
        z = that.z;
    }

    void assign(float _x, float _y, float _z)
    {
        x = _x;
        y = _y;
        z = _z;
    }

    float maxDistance(Point& that)
    {
        float dx = fabs(x - that.x);
        float dy = fabs(y - that.y);
        float dz = fabs(z - that.z);

        return max(dz, max(dx, dy));
    }
};

class Leg
{
public:

    Leg()
        : _debug(false)
        , _attached(false)
        , _cServoRestAngle(PI / 2)
        , _fServoRestAngle(PI / 2)
        , _tServoRestAngle(PI / 2)
        , _cServoDirection(0.0)
        , _fServoDirection(0.0)
        , _tServoDirection(0.0)
    {}

    void debug(bool on)
```

```
{
    _debug = on;
}

void attach(int coxaPin, int femurPin, int tibiaPin)
{
    _attached = true;

    if (_debug)
        return;

    _coxaPin = coxaPin;
    _femurPin = femurPin;
    _tibiaPin = tibiaPin;

    attach();
}

void attach()
{
    _cServo.attach(_coxaPin);
    _fServo.attach(_femurPin);
    _tServo.attach(_tibiaPin);
}

void detach()
{
    _cServo.detach();
    _fServo.detach();
    _tServo.detach();
}

void configureCoxa(float cStartX,
                   float cStartY,
                   float cStartAngle,
                   float cFemurOffset)
{
    _cStart.x = cStartX;
    _cStart.y = cStartY;
    _cStartAngle = cStartAngle;
    _cFemurOffset = cFemurOffset;
}

void configureFemur(float fStartZOffset,
```

```
                float fStartFarOffset,
                float fLength,
                float fStartAngle)
{
   _fStartZOffset = fStartZOffset;
   _fStartFarOffset = fStartFarOffset;
   _fLength = fLength;
   _fStartAngle = fStartAngle;
}

void configureTibia(float tLenght,
                float tStartAngle)
{
   _tLenght = tLenght;
   _tStartAngle = tStartAngle;
}

void configureServoDirections(float cServoDirection,
                        float fServoDirection,
                        float tServoDirection,
                        bool thirdQuarterFix)
{
   _cServoDirection = cServoDirection;
   _fServoDirection = fServoDirection;
   _tServoDirection = tServoDirection;
   _thirdQuarterFix = thirdQuarterFix;
}

void tuneRestAngles(float cServoRestAngle,
                float fServoRestAngle,
                float tServoRestAngle)
{
   _cServoRestAngle = cServoRestAngle;
   _fServoRestAngle = fServoRestAngle;
   _tServoRestAngle = tServoRestAngle;
}

bool reset()
{
   move(0, 0, 0);
}

bool reach(Point& dest)
{
```

```
            float hDist = sqrt( sqr(dest.x - _cStart.x) +  sqr(dest.y - _cStart.y) );
            float additionalCoxaAngle = hDist == 0.0 ? DONT_MOVE
                                        : asin( _cFemurOffset / hDist );

            float primaryCoxaAngle = polarAngle(dest.x - _cStart.x, dest.y -
_cStart.y, _thirdQuarterFix);

            float cAngle = hDist == 0.0 ? DONT_MOVE
                                : primaryCoxaAngle - additionalCoxaAngle -
_cStartAngle;

            float localDestX = hDist <= _cFemurOffset
               ? - _fStartFarOffset
               : sqrt(sqr(hDist) - sqr(_cFemurOffset)) - _fStartFarOffset;

            float localDestY = dest.z - _fStartZOffset;

            float localDistSqr = sqr(localDestX) + sqr(localDestY);
            if (localDistSqr > sqr(_fLength + _tLenght))
            {
               log("Can't reach!");
               return false;
            }

            float A = -2 * localDestX;
            float B = -2 * localDestY;
            float C = sqr(localDestX) + sqr(localDestY) + sqr(_fLength) -
sqr(_tLenght);
            float X0 = -A * C / (sqr(A) + sqr(B));
            float Y0 = -B * C / (sqr(A) + sqr(B));
            float D = sqrt( sqr(_fLength) - (sqr(C) / (sqr(A) + sqr(B))) );
            float mult = sqrt ( sqr(D) / (sqr(A) + sqr(B)));
            float ax, ay, bx, by;
            ax = X0 + B * mult;
            bx = X0 - B * mult;
            ay = Y0 - A * mult;
            by = Y0 + A * mult;
            float jointLocalX = (ax > bx) ? ax : bx;
            float jointLocalY = (ax > bx) ? ay : by;

            float primaryFemurAngle = polarAngle(jointLocalX, jointLocalY,
false);
            float fAngle = primaryFemurAngle - _fStartAngle;
```

```
            float primaryTibiaAngle = polarAngle(localDestX - jointLocalX,
localDestY - jointLocalY, false);
            float tAngle = (primaryTibiaAngle - fAngle) - _tStartAngle;

            move(cAngle, fAngle, tAngle);
        }

            void configureDefault(Point def, bool move)
        {
            _defaultPos = def;
            _currentPos = def;

            if (move)
                reach(def);
        }

        void reachRelativeToDefault(Point& dest)
        {
            _currentPos.assign(_defaultPos + dest);
            reach(_currentPos);
        }

        void reachRelativeToCurrent(Point& dest)
        {
            _currentPos.assign(_currentPos + dest);
            reach(_currentPos);
        }

        Point getCurrentRelative()
        {
            return _currentPos - _defaultPos;
        }

        Point& getCurrentPos()
        {
            return _currentPos;
        }

        Point& getDefaultPos()
        {
            return _defaultPos;
        }
```

```
private:

void move(float cAngle, float fAngle, float tAngle)
{
    if (_cServoDirection == 0.0 || _fServoDirection == 0.0 ||
_tServoDirection == 0.0)
        log("ERROR: Null servo directions detected");

    if (! _attached)
        return;

    int mc = rad2deg(_cServoDirection * cAngle + _cServoRestAngle);
    int mf = rad2deg(_fServoDirection * fAngle + _fServoRestAngle);
    int mt = rad2deg(_tServoDirection * tAngle + _tServoRestAngle);

    if (! _debug)
    {
        if (cAngle != DONT_MOVE)
            _cServo.write(coxaLimit(mc));
        if (fAngle != DONT_MOVE)
            _fServo.write(femurLimit(mf));
        if (tAngle != DONT_MOVE)
            _tServo.write(tibiaLimit(mt));
    }
    else
    {
        log("Angles:");
        log(mc);
        log(mf);
        log(mt);
    }
}

int coxaLimit(int x)
{
    const int cMin = 90 - 45;
    const int cMax = 90 + 45;

    if (x < cMin) return cMin;
    if (x > cMax) return cMax;

    return x;
}
```

```
    int femurLimit(int x)
    {
        const int fMin = 90 - 45;
        const int fMax = 90 + 45;

        if (x < fMin) return fMin;
        if (x > fMax) return fMax;

        return x;
    }

    int tibiaLimit(int x)
    {
        const int tMin = 90 - 45;
        const int tMax = 90 + 45;

        if (x < tMin) return tMin;
        if (x > tMax) return tMax;

        return x;
    }


private:
    bool _debug;

    Point _cStart
    float _cServoRestAngle;
    float _cServoDirection;
    float _cStartAngle;
    float _cFemurOffset;
    float _fStartZOffset;
    float _fStartFarOffset;
    float _fLength;
    float _fServoRestAngle;
    float _fServoDirection;
    float _fStartAngle;
    float _tLenght;
    float _tServoRestAngle;
    float _tServoDirection;
    float _tStartAngle;

    Point _defaultPos;
    Point _currentPos;
```

```
    bool _attached;
    bool _thirdQuarterFix;

    int _coxaPin;
    int _femurPin;
    int _tibiaPin;

    Servo _cServo;
    Servo _fServo;
    Servo _tServo;
};

#endif
```

ПРИЛОЖЕНИЕ В
Схема проекта