

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	9
---------------------------	----------

Глава 1 Обзор состояния вопроса и постановка задачи	
--	--

исследования	13
1.1 Степень изученности темы исследования	13
1.2 Основные принципы построения модульных систем управления технологическим оборудованием	15
1.3 Микросервисная архитектура как основа создания модульных систем управления технологическим оборудованием	17
1.4 Выводы к первой главе	19

Глава 2 Пользовательский графический интерфейс систем	
управления технологическим оборудованием	21

2.1 Виды пользовательского интерфейса систем управления технологическим оборудованием и его основные функции	21
2.2 Инструменты разработки графического интерфейса . . .	26
2.3 Структурно-модульный подход к разработке графического интерфейса	29
2.3.1 Требования к разработке графического интерфейса	29
2.3.2 Описание структурно-модульного подхода	30
2.3.3 Графический интерфейс системы ЧПУ на основе структурно-модульного подхода	33
2.4 Протокол взаимодействия пользовательского графического интерфейса	35
2.5 Отображение модуля графического интерфейса на эталонную модель RAMI 4.0	38
2.6 Выводы к второй главе	42

Глава 3 Реализация модуля графического интерфейса	44
3.1 Описание реализованного модуля графического интерфейса	44
3.2 Описание взаимодействия реализованного модуля графического интерфейса	47
3.3 Состав технических средств	49
3.4 Выводы к третьей главе	50
ЗАКЛЮЧЕНИЕ	52
ПЕРЕЧЕНЬ РИСУНКОВ	54
ПЕРЕЧЕНЬ ТАБЛИЦ	55
Список используемых сокращений	56
Список используемых источников	58
Приложение А Программная реализация обработчиков основных запросов на сервер	61
Приложение Б HTML реализация графической оболочки	64
Приложение В Графические формы	66
Приложение Г Микрокомпьютер ODROID-C2	69
Приложение Д Общий вид координатной платформы	70

ВВЕДЕНИЕ

Анализируя тенденции современного производства, стоит отметить одно наиболее важное изменение — переход от массового производства конвейерного типа к гибким автоматизированным и роботизированным производственным комплексам. Начало распространения комплексной автоматизации было обусловлено множеством технических предпосылок, и в первую очередь — это появление в начале 80-х годов достаточно мощных микропроцессоров и носителей памяти, а также удешевление и распространение компьютерной техники, что позволяло разрабатывать и внедрять такие системы. Естественно, одним из направлений автоматизации являлось совершенствование технологического оборудования. Благодаря научно-техническому прогрессу в этой области, удалось создать точное высокоточное оборудование с числовым программным управлением (ЧПУ) и привести его к такому виду, каким оно является сейчас.

В настоящее время информационные технологии распространяются повсеместно, и данный процесс не обходит стороной и производственную сферу. Фактически, промышленность вступает в новую эпоху Четвертой промышленной революции (известной также как Индустрия 4.0¹), которая влечет за собой массовое внедрение киберфизических систем на производстве. На предприятиях присутствует огромное количество задач разных уровней сложности, решение которых должно быть автоматизировано. Для этого применяется совокупность информационно-управляющих систем, иерархически связанных друг с другом: ERP системы, MES, SCADA системы и встроенные системы полевого² уровня. Однако не стоит забы-

¹Индустрия 4.0 — это понятие, введенное в Германии в 2011 году, обозначает собой государственную программу поддержки и развития промышленности, главной стратегией которой является интеграция киберфизических систем в производственные процессы.

²Полевое устройство (от англ. Field Device) — интеллектуальные датчики и актуаторы, имеющие возможность обрабатывать данные и взаимодействовать в сети [1].

вать, что технологическое оборудование, являясь одной из важнейших частей производства, представляет собой не менее сложную информационно-управляющую систему. И облегчение процесса внедрения оборудования в общую производственную сеть является одной из главных задач Индустрии 4.0.

На данный момент в России уже предпринимаются попытки решения данной проблемы, так, например, разрабатывается и применяется платформа промышленного интернета Winnum³ [2]. Она позволяет считывать и обрабатывать данные с разнообразных станков и подсоединять их к общей производственной сети с помощью дополнительного узла. Интеграция в сеть с использованием такого узла требует сложных настроек, что обусловлено существенными различиями в организации систем управления технологическим оборудованием различных производителей. Однако на этапе перехода интеграция является приемлемой как временная мера.

В наши дни система управления технологическим оборудованием представляет собой монолитную систему, части которой жестко интегрированы и неотъемлемы друг от друга. Все это вызывает необходимость создания дополнительного слоя управления над таким оборудованием для объединения его с современной киберфизической системой, что в свою очередь требует значительных финансовых и временных затрат. Очевидно, что необходим пересмотр самой парадигмы проектирования оборудования с ЧПУ. Нужно рассматривать любое новое оборудование с точки зрения возможности включения его в единую информационно-телекоммуникационную среду с использование открытого протокола.

Все вышеперечисленное указывает на **актуальность** рассматриваемой темы и доказывает необходимость как пересмотра парадигмы

³Winnum — это платформа промышленного интернета, главными задачами которой являются мониторинг, диагностика и оптимизация производственных процессов и оборудования.

построения ЧПУ систем, так и компонентов системы управления, в том числе и человеко-машинного интерфейса.

Научная новизна выполненной работы заключается в предложенном структурно-модульном подходе к разработке графических интерфейсов модульной системы управления технологическим оборудованием.

Практическая ценность работы заключается в следующем:

- Разработаны программные модули компонента графического интерфейса и библиотека графических элементов системы управления технологическим оборудованием.
- Разработанный программный компонент был развернут на аппаратной платформе ODROID-C2⁴.
- Налажено взаимодействие с ядром ЧПУ модульной системы управления технологическим оборудованием.

Объектом исследования является пользовательский интерфейс системы управления технологическим оборудованием. В качестве **предмета исследования** рассматриваются методики и инструменты, применимые для его создания.

Целью данной работы является обоснование применения структурно-модульного подхода к разработке графического интерфейса системы управления технологическим оборудованием. Для достижения поставленной цели потребовалось решить следующие основные **задачи**:

- Рассмотреть существующие методы и инструменты создания графического пользовательского интерфейса.
- Определить требования к компоненту графического интерфейса, предъявляемые модульной системой управления, определить роль,

⁴ODROID-C2 – 64-битный микрокомпьютер на базе ARM S905 Amlogic [3].

которую занимает данный компонент, в соответствии с выбранной архитектурой системы управления.

- Создать компонент пользовательского интерфейса и библиотеку базовых графических элементов.

Структура магистерской диссертации. Магистерская диссертация состоит из введения, трёх глав, заключения, библиографии и трёх приложений.

Во введении рассматриваются актуальность темы работы, степень её исследования, а также объект и предмет исследования. Также сформулированы цель и задачи работы.

В первой главе приведен обзор состояния исследуемой темы, определены основные принципы создания модульной системы управления технологическим оборудованием, рассмотрена предлагаемая архитектура модульной системы управления.

В второй главе определены основные функции пользовательского графического интерфейса в системах управления технологическим оборудованием, рассматриваются существующие методы и инструменты создания пользовательского графического интерфейса, проводится их сравнительный анализ. Обосновывается выбор протокола взаимодействия разрабатываемого модуля. Определяется роль и перечень требований к модулю графического интерфейса в реализуемой модульной системе управления технологическим оборудованием.

В третьей главе демонстрируются аспекты работы реализованного модуля пользовательского графического интерфейса.

В заключении приводятся результаты и выводы.

Глава 1

Обзор состояния вопроса и постановка задачи исследования

1.1 Степень изученности темы исследования

В последнее десятилетие стали активно проводиться работы по переходу к цифровому производству и интеллектуализации. Разработчики стремятся, чтобы вещи, которыми раньше управляли вручную или с помощью жестких автоматических систем, получали доступ в сеть и сами договаривались о выполняемых работах. Технологическое оборудование является одной из ключевых и дорогостоящих частей производства, и для того, чтобы интегрировать его в обновленную производственную и технологическую среду, необходимо произвести обширные как научные, так и инженерные исследования.

Например, исследователи из университетов Керетаро (Autonomous University of Queretaro) и Гуанахуато (University of Guanajuato) в Мексике работают над созданием мультиагентной платформы для управления технологическим оборудованием [4]. Данная система называется MADCON¹ и представляет собой платформу с открытой архитектурой, основанную на мультиагентных программно-аппаратных модулях. Цель разработчиков – создание такой платформы, которая бы удовлетворяла требованиям реконфигурируемости для интеллектуальных машин следующего поколения. Аппаратные модули данной системы базируются на программируемых вентильных матрицах (ППВМ), также известных как FPGA, а программные модули используют XML для описания функций системы и графического интерфейса.

¹MADCON — Multi-Agent Distributed Controller.

Аналогичные работы проводятся и в России. Например, ученые из Московского государственного технологического университета «СТАНКИН» предлагают подход к построению переносимого ядра ЧПУ на основе платформы независимых библиотек [5]. Открытая архитектура данной системы ЧПУ включает в себя уровни абстракции для реализации различных человеко-машинных интерфейсов, а также имеет возможность описания компонентов системы на различных языках программирования.

Компоненты системы связываются между собой по протоколам семейства Fieldbus², например, SERCOS [7, 8], EtherCAT [9], CAN-bus [10], Modbus [11] и т. д. Более того, специалисты университета самостоятельно разработали часть программно-аппаратных компонентов. Человеко-машинный интерфейс в данной системе обеспечивается тремя видами взаимодействия: с использованием физической стойки управления, удаленного устройства управления и веб-терминала, доступного с любого устройства с выходом в сеть.

В других странах также ведутся работы в этом направлении. В Хуажунском университете науки и технологии (Huazhong University of Science and Technology) в Китае разрабатывают платформу для создания открытых ЧПУ систем [12]. Основными её задачами являются упрощение и сопровождение разработки переиспользуемых модулей и интеграция их в прикладную систему управления. Кроме того, данная платформа включает инструменты моделирования и тестирования получаемых систем.

Кроме исследований, непосредственно связанных с созданием компонентов цифровых и интеллектуальных производств, ведутся работы по стандартизации. И лидирующую роль в этом вопросе занимают Германия и США. Немецкие специалисты разработали множество документов и стандартов, связанных с цифровым производством [13–15]. Основным

²Fieldbus – семейство протоколов промышленных сетей, используемых для распределенного контроля в реальном времени, описывается стандартом IEC 61158 [6].

является документ, посвященный эталонной архитектурной модели, также известной как RAMI 4.0 (Reference Architectural Model Industry 4.0) [16]. Параллельно в США были проведены аналогичные работы, однако стандарт был назван Industrial Internet Reference Architecture (IIRA) [17].

1.2 Основные принципы построения модульных систем управления технологическим оборудованием

Современные системы управления технологическим оборудованием отличаются комплексностью и монолитностью. Каждый их элемент – это «весь в себе» с жёсткой и иерархической архитектурой. Всё в подобных системах направлено на обеспечение качества, надежности и бесперебойной работы. Инертность подобных систем заставляет разработчиков автоматизированных систем управления технологическими процессами (АСУ ТП) следовать этому же принципу монолитности, потому что технологическое оборудование с централизованным управлением нельзя эффективно внедрить в децентрализованную производственную среду.

Вследствие этого, упор делается на интеграцию, то есть на объединение разрозненных компонентов в единую производственную систему, а не на интероперабельность – создание открытого интерфейса взаимодействия, позволяющего отдельным компонентам оставаться автономными, но способными общаться с другими компонентами в случае необходимости. Соответственно, желательно переходить к разработке оборудования с модульной архитектурой.

Подобная архитектура базируется на двух основных постулатах:

1. **Унификация.** Под унификацией понимается открытая программно-аппаратная структура, позволяющая создавать новые типы оборудо-

дования и программного обеспечения по принципу «интеллектуального конструктора». Унификация достигается за счет разбиения единого изделия на крупные взаимозаменяемые блоки с четким описанием входных и выходных параметров каждого блока.

2. *Гибридизация*. Позволяет создавать установки, являющиеся совокупностью уже существующих. Например, можно сконфигурировать аддитивно-субстрективную машину для быстрого прототипирования, сочетающую в себе характеристики 3D-принтера и трехкоординатного фрезерного станка или совместить фрезерную головку для черновой обработки заготовок с лазером для полирования определенных поверхностей.

Применяя данные принципы, в первую очередь необходимо выделить основные части технологического оборудования:

- рабочий орган;
- координатное шасси, перемещающее рабочий орган в пространстве;
- блок числового программного управления.

Очевидно, что координатное шасси является наиболее универсальным блоком, на который могут быть установлены различные рабочие органы, за счет чего может быть изменен тип оборудования. Например, сменив фрезерную головку на лазерную, можно сделать из фрезерного станка гравировальный или форматно-раскроечный, а поставив экструдер для пластика – 3D-принтер.

Однако не стоит забывать о том, что смена рабочего органа — это не только механическое изменение параметров оборудования, но и смена алгоритма управления. Блок числового программного управления должен заранее знать о каждом рабочем органе, который может быть установлен

на шасси. Очевидно, что при таком подходе система остается монолитной с иерархическим управлением. Поэтому необходимо выделить базовую часть архитектуры, определить спецификацию протокола взаимодействия и создания новых программно-аппаратных модулей, которые могут быть динамически включены в систему. К базовой части относится алгоритм управления электрическими приводами координатного шасси, а все остальное является внешними компонентами.

Подобный подход полностью изменяет стиль управления производственной средой, позволяя добиться бесшовного объединения различных производственных установок в единую сеть.

1.3 Микросервисная архитектура как основа создания модульных систем управления технологическим оборудованием

Сервис-ориентированная архитектура³ является быстро развивающейся концепцией, все чаще используемой для реализации распределенных программных систем. Основа подобной архитектуры – набор слабо связанных заменяемых компонентов, оснащенных унифицированными интерфейсами для взаимодействия по стандартизованным протоколам. Первые упоминания о концепции SOA можно встретить в литературе начиная с 2009 года [18, 19].

Микросервисная архитектура⁴ является современной интерпретацией SOA (первые упоминания об MSA относятся к 2012 году, хотя этот термин употреблялся и ранее), используемой для создания децентрализованного программного обеспечения [20]. В данной архитектуре под сервисом понимается процесс, выполняемый операционной системой, кото-

³В англоязычной литературе обозначается как Service-oriented architecture (SOA).

⁴В англоязычной литературе обозначается как Microservices architecture (MSA).

рый взаимодействует с другими процессами по сетевому интерфейсу для достижения общей цели. Микросервисы могут быть построены с применением любого языка или фреймворка, но должны использовать общий протокол, который позволяет скрыть особенности реализации каждого из микросервисов. При этом каждый микросервис работает лишь с небольшой, максимально ограниченной областью задач, выполняя минимум функций.

На сегодняшний день не существует четкого определения MSA. Тем не менее, из всего многообразия встречающихся в литературе формулировок попытаемся выделить те особенности данного подхода, которые особенно важны при проектировании распределенных систем ЧПУ:

- 1. Архитектура микросервисов позволяет легко заменять модули, входящие в состав системы.* В качестве примера можно представить человеко-машинный интерфейс для взаимодействия с оборудованием. Классический интерфейс – это набор физических элементов управления: индикаторов, кнопок, переключателей и т. д. Подобный интерфейс очень удобен в массовом производстве, когда оборудование используется в составе конвейерной линии. Однако в условиях промышленной лаборатории (FabLab), подобный физический способ взаимодействия может быть избыточным. Использование микросервисного подхода дает возможность легко заменить физическое устройство управления на виртуальное, что позволит управлять оборудованием, например, через веб-интерфейс с любого устройства, подключенного к сети.
- 2. Все модули организованы вокруг функций.* Архитектура микросервисов позволяет разделять функционал каждого блока. Рассмотрим в качестве примера фрезерную головку. Данный модуль состоит из физической части: двигатель, патрон, контроллер, датчики и логической – модуль управления. Физически обе части рас-

положены в едином блоке. Но с точки зрения архитектуры – это два разных сервиса. Один из них отвечает за низкоуровневые команды управления приводом головки, сбором данных от датчиков и т. д. А другой – это высокоуровневый интерфейс пользователя, который может включать в себя доступ к элементам управления, заданием технологических параметров, и даже включать в себя систему подготовки управляющих программ, нацеленных именно на фрезерную обработку. Каждый из них отвечает строго за свою функцию системы и ничего не знает о реализации других микросервисов.

3. *Каждый микросервис является эластичным, легко модифицируемым, но при этом законченным программным продуктом.* Данное утверждение подразумевает, что при разработке системы ЧПУ необходимо придерживаться принципа увеличения связности и уменьшения связанности. Это позволяет с одной стороны сфокусироваться на разработке и отладке каждого отдельного блока, а с другой – дает возможность упростить методику добавления и модификации функций системы в целом.

Микросервисная архитектура является современным и быстроразвивающимся направлением развития кибер-физических производственных систем и может быть успешно использована для создания распределенных систем управления технологическим оборудованием.

1.4 Выводы к первой главе

1. Проведенные исследования показали, что предметная область активно развивается в направлении повсеместного внедрения информационных и коммуникационных технологий, что сопровождается увеличением автономности и независимости компонентов. Это вле-

чет за собой изменение парадигмы управления как производством в целом, так и отдельно взятыми компонентами, в сторону модульности. Соответственно, данное высказывание справедливо и для систем управления технологическим оборудованием.

2. Основной проблемой систем управления технологическим оборудованием является монолитность их архитектуры. Для реализации современной системы управления технологическим оборудованием, требуется делать упор на интероперабельность – создание открытого интерфейса взаимодействия.
3. Проведя исследование микросервисного подхода, можно сделать вывод, что данный подход применим к системам ЧПУ и соответствует тенденциям развития промышленности, обусловленным четвёртой промышленной революцией.

Основной задачей представленного исследования является установление необходимости и состоятельности структурно-модульного подхода для реализации пользовательского графического интерфейса микросервисной модульной системы управления технологическим оборудованием.

Глава 2

Пользовательский графический интерфейс систем управления технологическим оборудованием

2.1 Виды пользовательского интерфейса систем управления технологическим оборудованием и его основные функции

Любая система управления технологическим оборудованием состоит из трех основных частей:

- ядро системы управления;
- программируемый логический контроллер;
- человеко-машинный интерфейс.

Обычно человеко-машинный интерфейс в технологическом оборудовании реализуется двумя способами: *аппаратным* и *программным*.

Классическая реализация аппаратной стойки управления включает в себя множество кнопок, регуляторов и других физических элементов ([рис. 2.1](#)). При этом все чаще стали получать распространение стойки, которые имеют в своем составе сенсорные экраны ([рис. 2.2](#)), позволяющие в разы сократить количество физических элементов управления, оставив только самые критически важные, например, кнопку аварийного останова.

Одним из главных недостатков таких стоек является то, что они, как правило, располагаются непосредственно рядом с технологическим оборудованием и не обеспечивают возможность удаленного управления.



Рисунок 2.1 — Стойка FANUC Series 0i-TD



Рисунок 2.2 — Стойка KSE-TOUCH CNC

Многие производители технологического оборудования, пытаясь решить эту проблему, прилагают программное обеспечение, установив которое на персональный компьютер, появляется возможность осуществлять удаленное управление. Однако с таким подходом сопряжен *ряд проблем*.

Во-первых, обычно поддерживается только ряд известных платформ, что вносит ограничения в организацию управления оборудованием.

Во-вторых, технологическое оборудование имеет долгий срок службы, а учитывая скорость развития информационных технологий и компьютерной техники, такое программное обеспечение придется регулярно обновлять. При этом процесс обновления так или иначе будет требовать определенного набора действий от пользователя, поскольку изменения будут проходить именно на используемой целевой машине.

В-третьих, отсутствие мобильности, поскольку оператор все также остается закрепленным за определённым рабочим местом.

Вместе с тем, учитывая тенденции развития современных производственных систем, важно не только обеспечить удаленное управление, но и определенный уровень абстракции. Это позволило бы незаметно для нижнего уровня осуществлять управление и наблюдение как с помощью оператора, так и с использованием другой программной системы более высокого порядка.

Функции пользовательского интерфейса обычно подразделяются на *пять групп* [21]:

1. *Операционные функции*. Они используются часто и помогают управлять технологическим оборудованием, а также показывают его состояние. Они занимаются отображением позиции, расстояния, подачи по каждой оси, скорости шпинделя, выполняемого блока программы и статуса. Кроме того, предусмотрены функции, помогающие работать с технологическим оборудованием, такие как переме-

щение в режимах Jog¹ и MDI², поиск программ, редактор программ и управление инструментами.

2. *Функции настройки параметров.* В системе ЧПУ имеются различные параметры для внутреннего использования, которые подразделяются на три вида:

- параметры станка, которые используются для настройки станка, сервопривода, смещения инструмента, рабочей координаты;
- параметры программы, которые должны быть установлены при редактировании программы обработки детали;
- параметры настройки, которые используются, чтобы адаптировать машину к требованиям пользователя.

Эти функции предоставляют интерфейс для установки, хранения и поиска параметров.

3. *Функции редактирования программы.* Эти функции позволяют редактировать и модифицировать программу обработки детали, которая представляет собой G-код.

4. *Функции контроля и сигнализации.* Система ЧПУ всегда уведомляет пользователя о состоянии и, при необходимости, запускает на исполнение важные задачи и информирует пользователя о результате. Функции сигнализации нештатной ситуации, методе аварийного восстановления и т. д.

5. *Служебные функции.* Помимо основных четырёх групп, представляется множество полезных функций для оказания помощи

¹Jog – режим, который может быть использован для перемещения по заданной оси с заданной скоростью.

²MDI (Manual data input) – режим ввода команды (например, G-код) для прямого выполнения.

пользователям. Например, осуществление сетевого взаимодействия и управление данными пользователя.

Графический интерфейс в первую очередь предоставляет широкий набор возможностей по визуализации процесса обработки. Визуализация процесса обработки является важной, поскольку дает возможность пользователю оборудования определить состояние и этап процесса обработки в реальном времени. В технологическом оборудовании визуализация обычно осуществляется несколькими способами:

- Экран, отображающий проделанный путь рабочей головки. Он позволяет визуально оценить получаемый результат. Особенно полезен в случаях, когда нет возможности наблюдения происходящей обработки с камер или в реальности, например, при лазерной обработке или сварке.
- Экран с выполняемой управляющей программой позволяет точно определить этап проводимых работ и текущую команду.
- Набор экранов, отображающих текущие координаты рабочей головки, текущие настройки, инструмент, скорости и т. д.
- Экран видеонаблюдения. Применяется, когда оборудование оснащено камерами, обычно когда нет прямого доступа для наблюдения рабочей области.
- Экран с 3D-моделью выполняемого процесса в реальном времени.

Из всего вышесказанного следует, что графическая часть пользовательского интерфейса позволяет наиболее полно и наглядно обеспечивать связь технологического оборудования с пользователем, используя современные технологии, например, 3D-моделирование и работу с видео.

2.2 Инструменты разработки графического интерфейса

Инструменты разработки графического интерфейса делятся на три основные категории:

- низкоуровневые фреймворки;
- высокоуровневые фреймворки;
- фреймворки для разработки интерфейса веб-приложений.

Низкоуровневые фреймворки решают задачи отрисовки окон, движения окон, взаимодействие с клавиатурой и мышью. В зависимости от типа операционной системы они либо интегрированы в операционную систему, либо работают поверх неё. Разработчики графических интерфейсов редко работают с ними напрямую и используют высокоуровневые фреймворки для решения своих задач. Однако необходимо понимать, что базой любого графического интерфейса являются именно такие низкоуровневые компоненты. В качестве примеров можно привести Cocoa, Windows API и X Window System.

Высокоуровневые фреймворки решают задачи, связанные непосредственно с разработкой приложений с графическим интерфейсом, они включают в себя наборы примитивов графического интерфейса, инструменты для работы с ними и некоторые дополнительные возможности. Высокоуровневые фреймворки разделяются на платформо-зависимые и кроссплатформенные.

Платформо-зависимые фреймворки позволяют создавать приложения с графическим интерфейсом, которые поддерживаются только определенной операционной системой. Несмотря на этот недостаток, можно выделить ряд плюсов таких фреймворков. Во-первых, использование

нативного кода, а соответственно, доступ ко всем возможностям операционной системы и в теории ускорение работы. Во-вторых, соответствие элементов графического интерфейса стилем темам, установленным разработчиками операционной системы. Примерами таких фреймворков являются MacApp, Microsoft Foundation Classes и MUI³.

Кроссплатформенные фреймворки позволяют создавать программы с графическим интерфейсом под множество платформ без изменения программного кода. Многие из них позволяют автоматически изменять вид элементов графического интерфейса во время разработки и в зависимости от платформы. Одними из самых популярных являются фреймворки Qt и GTK+.

Тем не менее, сейчас для создания приложений с графическим интерфейсом все чаще используются веб-технологии. Главными причинами являются не только широкие возможности создания собственных элементов и виджетов со сложной логикой, но и распространение сетей повсеместно. Использование веб-технологий при создании интерфейсов позволяет обеспечить удаленный доступ, мобильность, мультиплатформенность и экономию ресурсов пользователя.

В настоящее время имеется значительное количество разнообразных *фреймворков для разработки интерфейса веб-приложений*. Однако в действительности все они сводятся к использованию стандартного стека технологий HTML/CSS/JS. Данные фреймворки можно поделить на три вида:

- фреймворки, работающие по принципу конструктора;
- JS-фреймворки;
- Специальные языки программирования, компилирующиеся в JS.

³MUI (Magic User Interface) – фреймворк на операционных системах AmigaOS и MorphOS.

Bootstrap является самым широко используемым фреймворком, работающим по принципу конструктора. Он состоит из набора готовых стилизованных шаблонов оформления, зачастую инкапсулирующих определенную логику. Более того, он включает в себя разнообразные инструменты создания элементов навигации, медиа-контейнеры, таблицы, сетки и типографику. Подключение осуществляется через базовый html файл так, как показано на [листинге 2.1](#).

Листинг 2.1 – Подключение фреймворка Bootstrap

```
1 <link href="{{ url_for('static',
2                         filename='dist/css/bootstrap.min.css')
3                     }}" rel="stylesheet">
4 <script src="{{url_for('static',
5                         filename = 'dist/js/bootstrap.min.js')
6                     }}>
7 </script>
```

ReactJS является известным представителем JS-фреймворков. Дан-ный фреймворк работает на уровне представления и результатом его работы является HTML. Другими словами, фреймворк не занимается описанием внешнего вида элементов, он осуществляет управление отри-сивкой, а также позволяет создавать связки HTML-JS. Вместо разделения функциональности и представления ReactJS позволяет создать набор компонентов. Создание компонентов происходит на языке JSX, который затем компилируется в JS. Подключение ReactJS указано в [листинге 2.2](#).

Листинг 2.2 – Подключение ReactJS

```
1 <script src="http://fb.me/react-0.10.0.min.js">
2 </script>
```

Язык **Elm** – представитель множества языков, которые компили-руются в JS, HTML и CSS. Он является функциональным и позволяет создавать графический интерфейс веб-приложений в стиле функциональ-ной парадигмы. Elm обладает собственной экосистемой, включающей компилятор, пакетный менеджер и систему сборки.

2.3 Структурно-модульный подход к разработке графического интерфейса

2.3.1 Требования к разработке графического интерфейса

Очевидно, что монолитный подход к разработке графического интерфейса неприемлем для модульной системы управления технологическим оборудованием. Причиной является появление динамической составляющей графического интерфейса. Динамическая составляющая означает, что наборы виджетов, предоставляемых пользователю, должны меняться и расширяться в зависимости от модулей (микросервисов), доступных в распределенной сети системы ЧПУ.

Кроме того, учитывая современные тенденции развития промышленности, выдвигается ряд требований:

1. *Обеспечить удаленный доступ* к управляемому технологическому оборудованию, учитывая мобильность и мультиплатформенность.
2. *Обеспечить конфигурируемость и расширяемость*, т. е. возможность динамически дополнять и заменять элементы пользовательского интерфейса в зависимости от установленного модуля или желания пользователя.
3. *Обеспечить безопасность*, в том числе, разграничение доступа для пользователей и других сервисов.
4. *Обеспечить интероперабельность*, т. е. использование открытого протокола. В качестве примера можно привести ситуацию, когда производственный процесс строится по принципу «переговоров» его участников. Для осуществления такого подхода система управления конкретного технологического оборудования должна быть расширена системой планирования. Одним из вариантов реализации

может быть создание системы поверх модуля пользовательского интерфейса с использованием открытого протокола. Это означает, что данная система использовала бы те же интерфейсы, которыми пользуется оператор, но без их графической составляющей, например, интерфейсом загрузки управляющей программы.

Учитывая вышеприведенные требования, предлагается использование структурно-модульного подхода.

2.3.2 Описание структурно-модульного подхода

Структурно-модульный подход – это гибридный подход, сочетающий в себе принципы как структурного, так и модульного подходов к разработке. Понятие модульного подхода появилось достаточно давно и сейчас лежит в основе многих других подходов. *К основным принципам модульного подхода* относятся следующие [22]:

- *Принцип изоляции определяемого понятия.* Модуль представляет собой чёрный ящик, инкапсулируя в себе определенное понятие и его реализацию.
- *Принцип создания проблемно-ориентированного контекста.* Благодаря декомпозиции на модули разработчик может частично формализовать проблемную часть предметной области. За счет этого получается чётко сформулировать перечень задач, решаемых конкретной системой.
- *Принцип заменяемости и локализации машинной зависимости программ.* Модульность позволяет системе быть реконфигурируемой и изменяемой, заменяя части системы без влияния на неё в целом. Это в свою очередь влечет аппаратную независимость программной системы.

- *Принцип представления модуля как абстрактного объекта.* Дан-
ный принцип позволяет определить некоторую семантику на опре-
дленном уровне абстракции независимо от используемого алгоритма,
типа данных, способа работы с памятью и т. д. Фактически это один
из главных принципов, который затем лёг в основу объектно-ориен-
тированного подхода.
- *Принцип поэтапной и параллельной разработки программ.* Обычно
разработка программ включает в себя несколько этапов [23]: анализ
требований к проекту, проектирование, реализация, тестирование
продукта, внедрение и поддержка. Под принципом поэтапной разра-
ботки понимается соответствие каждому этапу степени детализации
описания модуля. При этом модули могут создаваться разными
разработчиками параллельно.

Структурный подход базируется на основе способа проектирова-
ния «сверху-вниз», используя декомпозицию, но при этом сохраняя целост-
ность системы. Такой подход строится на следующих принципах [24]:

- *Принцип «разделяй и властвуй».* Принцип, согласно которому для
решения задачи производится ее деление на более мелкие до того
момента, пока задачи не станут элементарными. А затем, на основе
решений мелких задач, приходят к решению главной.
- *Принцип иерархического упорядочивания.* Данный принцип ука-
зывает, что организация системы должна быть уровневой (или
послойной).
- *Принцип абстрагирования.* Принцип подразумевает отделение наи-
более важных аспектов системы на определенном уровне от менее
важных с целью последовательной детализации функций системы.

- *Принцип непротиворечивости.* Заключается в поддержании согласованности элементов программной системы.
- *Принцип структурирования и независимости данных.* Данные должны быть структурированы, а также не должны зависеть от способа их передачи и обработки.

В результате комбинации двух вышеуказанных подходов был получен ***структурно-модульный подход***. Как можно заметить, модульный подход отличается обособленностью частей программного обеспечения друг от друга, которые были получены посредством декомпозиции. В то время как структурный подход наоборот определяет строгие иерархические связи между ними. Вместе с этим, главная идея структурно-модульный подхода – динамическое образование структур из модулей.

С одной стороны, производится декомпозиция на отдельные независимые друг от друга модули. При этом полученные модули соответствуют всем принципам модульного подхода, однако дополнительно добавляется принцип интероперабельности. Что означает, что данные модули должны иметь возможность коммуникации в распределенной сети по открытому интерфейсу. Открытым интерфейсом называется такой способ взаимодействия, который могли бы использовать любые модули вне зависимости от расположения, внутренних особенностей реализации и языка, на котором они написаны.

С другой стороны, дополнительно реализуется агрегатор ([рис. 2.3](#)). Данный компонент, используя открытый интерфейс модулей, динамически объединяет их в структуры. Вместо системы с плоской организацией создается система с иерархической организацией ([рис. 2.4](#)). Тем не менее, в данном подходе иерархия не является жесткой, она динамически адаптируется и изменяется в зависимости от задач и потребностей вышестоящего уровня.

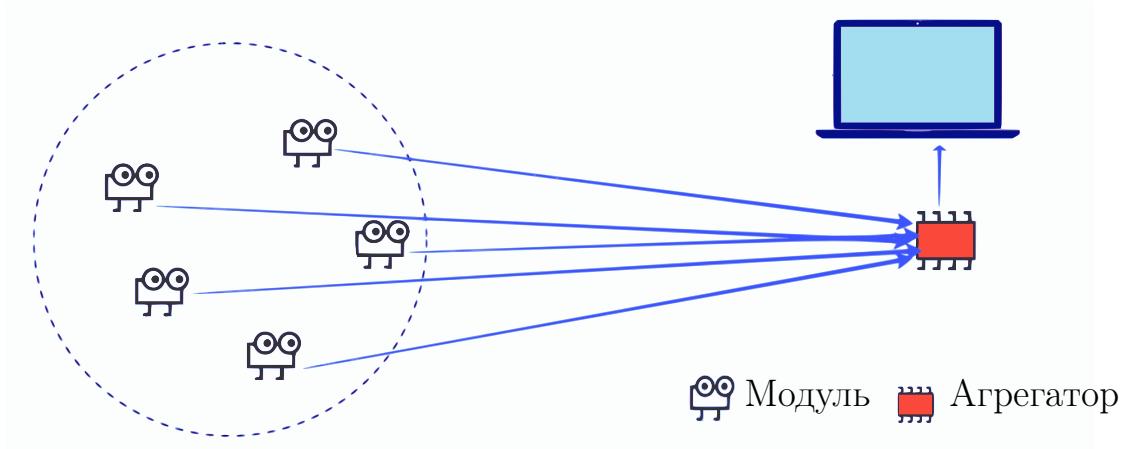


Рисунок 2.3 — Агрегатор

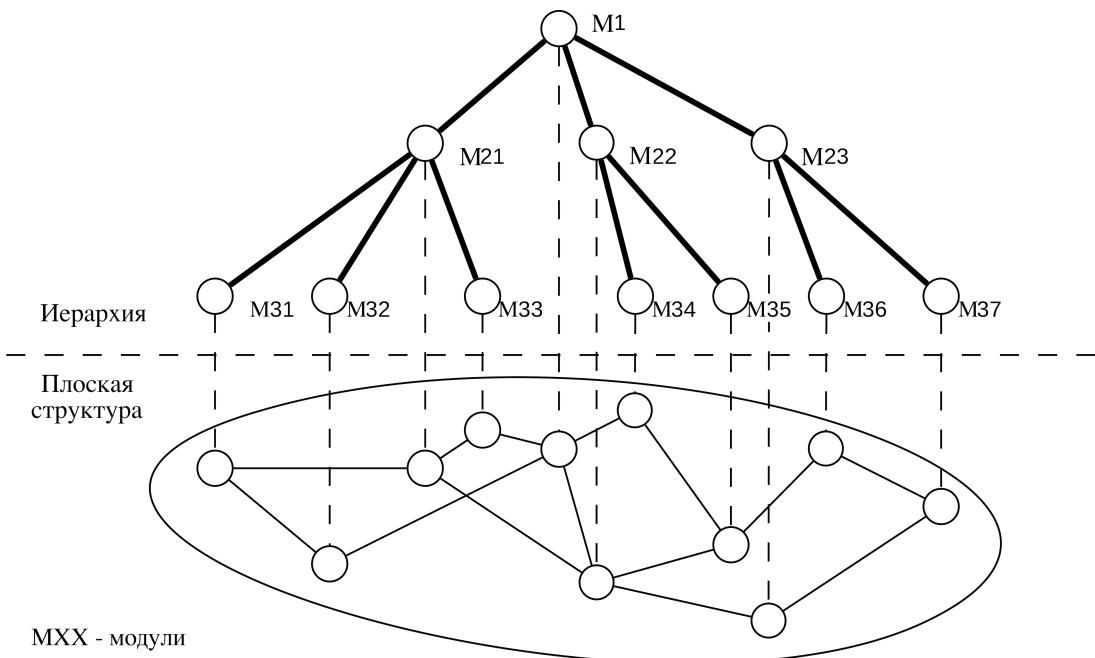


Рисунок 2.4 — Структура системы с иерархической организацией

2.3.3 Графический интерфейс модульной системы ЧПУ на основе структурно-модульного подхода

Система управления модульным технологическим оборудованием представляет собой гетерогенную распределенную сеть, узлами которой являются разнообразные модули системы, как программные, так и программно-аппаратные (рис. 2.5).

Таким образом, при создании компонента графического интерфейса необходимо учитывать возможность замены одного модуля

системы другим. Чтобы реализовать это, необходимо применить структурно-модульный подход.

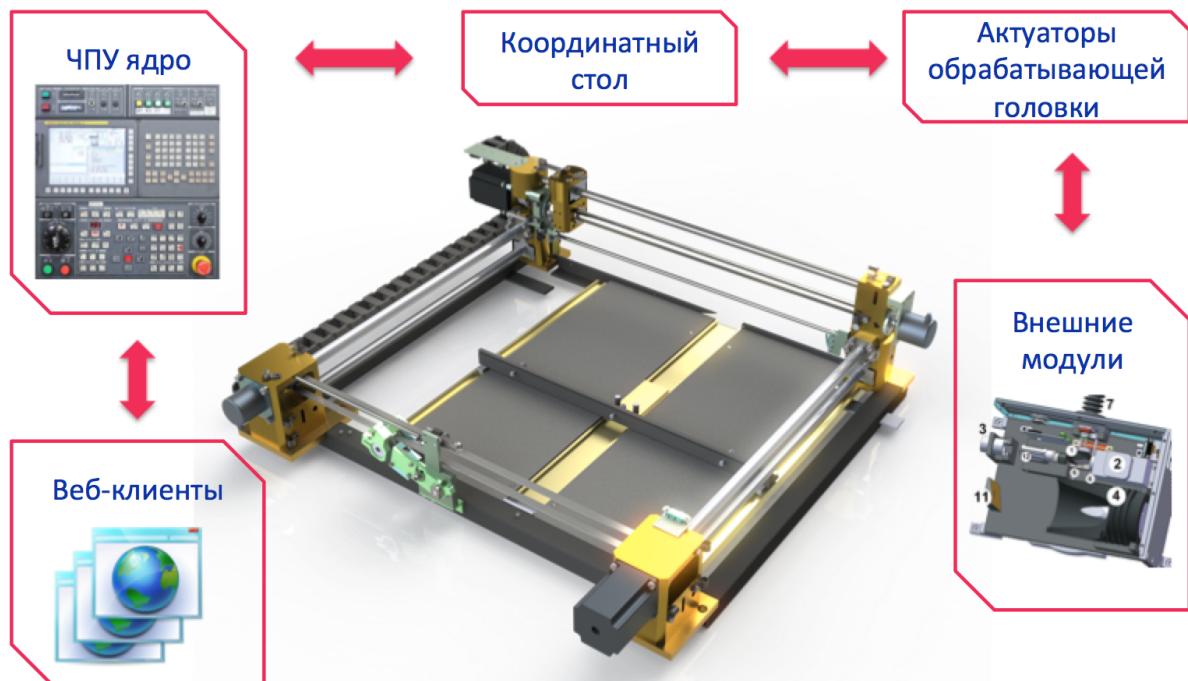


Рисунок 2.5 — Составляющие модульного оборудования с ЧПУ

Очевидно, что первичная декомпозиция графического интерфейса на модули соответствует перечню аппаратных модулей системы. При этом каждый такой модуль графического интерфейса (модуль ГИ) – самостоятельная автономная программная единица, которая находится на соответствующей аппаратной платформе и обладает свойством интероперабельности. Дальнейшая декомпозиция производится по функциям конкретного модуля системы, образуя набор виджетов (рис. 2.6). Виджеты для локального модуля графического интерфейса выступают в качестве данных и передаются другим участнику сети по запросу.

Для образования временной структуры используется агрегатор интерфейсов. Полученные модули графического интерфейса он компонует в базовую оболочку системы управления, где такая иерархия описывается корневым html файлом. Графические интерфейсы каждого модуля также встраиваются в этот корневой файл. В данном корневом файле

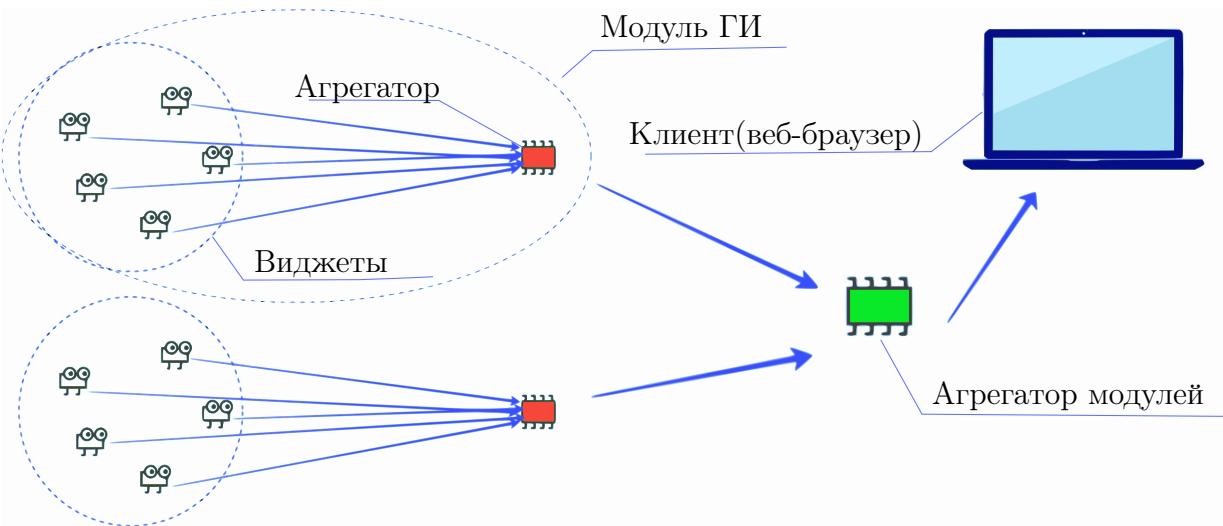


Рисунок 2.6 — Организация компонента графического интерфейса

графические интерфейсы распределены по категориям и доступны для пользователя. Сама структура может меняться двумя способами:

- Автоматически, в случае, если был заменен программно-аппаратный модуль системы ЧПУ.
- Вручную, когда пользователь самостоятельно выбрал необходимый набор графических форм под текущую задачу. Данную возможность должна обеспечивать базовая оболочка.

Из вышесказанного становится ясно, что агрегатор должен быть реализован в виде веб-сервера. Такой подход позволяет удовлетворить требование обеспечения удаленного доступа и мультиплатформенности. А также позволяет решить проблемы безопасности, используя стандартные способы, применяемые при веб-разработке.

2.4 Протокол взаимодействия пользовательского графического интерфейса

Микросервисная архитектура модульной системы управления предполагает применение общего облегченного протокола взаимодействия,

позволяющего скрыть особенности реализации модулей. Так как модуль пользовательского графического интерфейса является частью децентрализованной гетерогенной сети, необходимо использовать «умные» приемники сообщений и «глупые» каналы передачи данных. Подобное решение дает возможность делать модули системы максимально независимыми и сфокусированными на решении одной конкретной задачи.

Очевидно, что при работе в сетях TCP/IP, наиболее очевидным способом передачи данных являются «сырые» сокеты. Через этот программный интерфейс можно передавать байтовый поток данных. Но в разрабатываемой системе все модули, в том числе и модуль пользовательского графического интерфейса, обмениваются уже структурированной информацией. Поэтому более правильным подходом будет использование разновидности пакетной передачи данных. Каждый пакет – есть некоторое сообщение, которое может быть передано от одного модуля к другому. Получив данное сообщение, принимающий модуль должен подтвердить получение (в случае, если его содержимое ему понятно и может быть обработано) либо отклонить сообщение.

Легко заметить, что подобные сообщения очень похожи на те, которые используются в таких протоколах как XML-RPC, SOAP, BPEL или WSDL, но с одним важным отличием. Все эти протоколы используются в крупных промышленных приложениях и явно не подходят для встраиваемых систем. А так как модуль графического интерфейса должен осуществлять взаимодействие и со встраиваемыми системами, то данный недостаток является существенным. Для создания легковесной, открытой и расширяемой системы управления необходимо минимизировать слой абстракции, максимально используя преимущества низкоуровневых сокетов в сочетании с возможностью осуществлять маршрутизацию сообщений.

Таким образом, наиболее подходящим способом взаимодействиями

модулей проектируемой системы числового программного управления являются *очереди сообщений*.

Очередь сообщений является асинхронным протоколом передачи данных, то есть отправитель и получатель сообщения не взаимодействуют друг с другом напрямую, а только через очередь.

Можно выделить следующие *основные особенности очередей сообщений*, которые позволяют сделать вывод о том, что для проектируемой системы управления данный способ передачи данных является наиболее целесообразным:

- Очереди сообщений позволяют компонентам системы оставаться максимально независимыми друг от друга, исключая возможные взаимные блокировки, когда один из участников взаимодействия вынужден ожидать освобождения сетевых ресурсов, необходимых ему для передачи или приема данных.
- Очереди сообщений дают возможность экономить вычислительные ресурсы за счет отсутствия необходимости в сетевых буферах, в которых хранятся еще не переданные или еще не обработанные данные. По сути, очередь сообщений сама является универсальным сетевым буфером, не привязанным к какому-то конкретному узлу или процессу.
- Очереди сообщений легко масштабируются как по объему передаваемых данных, так и по пропускной способности.
- Очереди сообщений позволяют легко сглаживать пиковые нагрузки, возникающие в сети. Для проектируемой системы управления это особенно важно, так как в ней возможно смешение разных типов трафика. В частности, можно выделить как минимум высокоприоритетный трафик (данные от датчиков, команды управления) и низ-

коприоритетный трафик (загрузка исходного текста программы управления в контроллер, передача видео из зоны обработки и т. д.).

- Очереди сообщений позволяют существенно повысить отказоустойчивость системы, ведь сообщения остаются в очереди и могут быть обработаны даже в случае отказа отправившего их узла. Это можно продемонстрировать на примере передачи управления управляющей программы для оборудования с ЧПУ. Управляющая программа создается оператором в блоке интерфейса, а затем передается в блок исполнения в виде потока текстовых команд. Очевидно, что программа будет передана в виде нескольких сообщений, помещаемых в очередь, причем на максимально возможной скорости. Предположим, что какой-то момент блок интерфейса «зависает», данный факт фиксируется сторожевым таймером, который инициирует перезагрузку операционной системы. Однако во время перезагрузки блок исполнения продолжает считывать и обрабатывать сообщения из очереди, и для него процесс передачи данных не прерывается.
- Очереди сообщений гарантируют доставку сообщения, по крайней мере до тех пор, пока в сети функционирует хотя бы один узел, который может обработать их.
- Очереди сообщений не нарушают порядок передаваемых сообщений. Как правило, сообщения будут получены именно в том порядке, в котором они были отправлены.

2.5 Отображение модуля графического интерфейса на эталонную модель RAMI 4.0

Индустрия 4.0 включает в себя множество аспектов, и для того, чтобы определить общее виденье этих аспектов и их связь, была создана

эталонная модель RAMI 4.0 (рис. 2.7). Данная модель является иерархической и трехмерной, и по трём осям отображает предметную область с точки зрения информационных технологий, жизненного цикла изделий и организации производства.

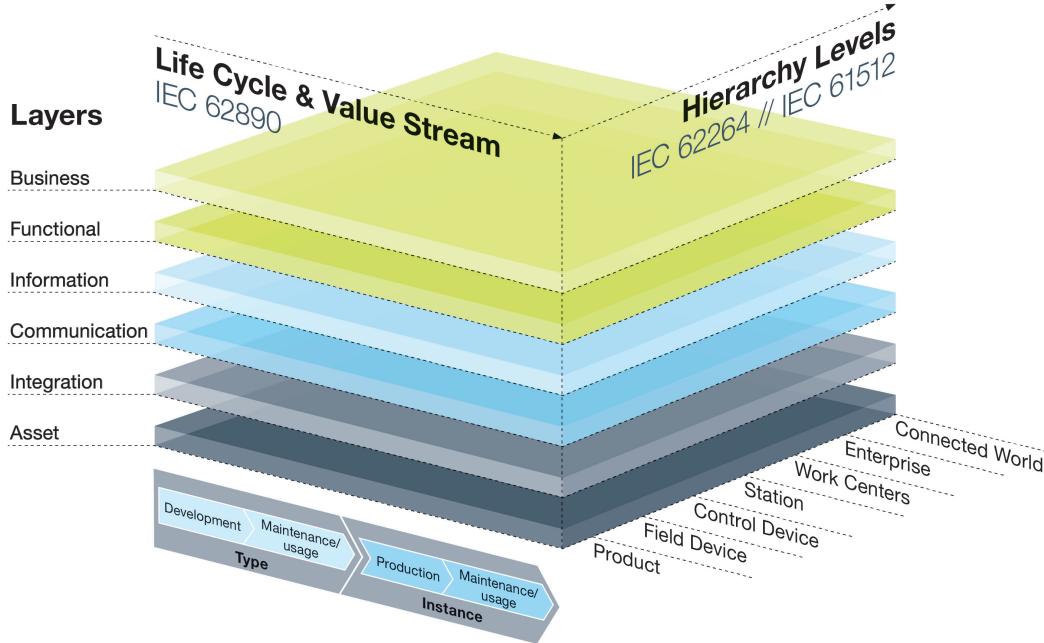


Рисунок 2.7 — Архитектурная модель RAMI 4.0 [http://winfwiki.wi-fom.de/index.php/Referenzplattform_Industrie_4.0]

В сфере информационных технологий использование таких типов моделей не редкость. Известный пример – иерархическая семиуровневая модель OSI [25], которая является эталонной моделью протоколов взаимодействия. Использование таких моделей позволяет достичь четкого понимания функций каждого уровня, а также определить интерфейсы связи между ними.

Одна из основных задач данной модели – обеспечить простое, наглядное и структурированное представление, которое бы помогало при создании описания компонентов и локализировало бы их роль в предметной области.

Таким образом, для того чтобы обосновать применение структурно-модульного подхода к реализации графических интерфейсов, далее будет представлено отображение на архитектурную модель RAMI 4.0 модуля

пользовательского графического интерфейса, разработанного с использованием вышеуказанного подхода.

Модуль графического интерфейса входит в состав системы управления технологическим оборудованием, поэтому на правой горизонтальной оси он находится в областях «устройство управления» и «станция». Модуль не входит в область «полевые устройства», поскольку не имеет связи с ними напрямую.

Наибольший интерес в данном случае представляет вертикальная ось. Модуль графического интерфейса находится на уровнях «интеграция», «взаимодействие», «информация» и «функционал». Вместе с этим, он обеспечивает открытые связи с вышестоящим и нижестоящим уровнем.

Уровень интеграции. Цель данного уровня – преобразование событий реального мира в события виртуального мира. На этом уровне рассматриваемый модуль обеспечивает преобразование действий пользователя (нажатие на кнопки, ввод программы и т. д.) в события в виртуальном мире. При этом пользователь находится на нижестоящем уровне – уровне ресурсов. Фактически, интеграцией занимается клиентская часть рассматриваемого модуля – т. е. браузер и JavaScript обработчики событий, активирующиеся при взаимодействии пользователя с графическими формами.

С другой стороны, модуль графического интерфейса входит в распределенную гетерогенную сеть системы управления технологическим оборудованием. Поэтому преобразование происходит и в обратную сторону. Например, сигнал окончания обработки передается на модуль графического интерфейса, а тот в свою очередь извещает об этом пользователя.

Уровень взаимодействия. Целью уровня взаимодействия является определение способа коммуникации, т. е. интерфейса соединения, протокола и формата данных, а также обеспечение связи уровня интеграции и информационного уровня. Рассматриваемый модуль имеет несколько интерфейсов соединения – беспроводное WiFi соединение,

Ethernet и последовательное соединение через UART. В качестве протокола используются очереди сообщений, реализованные с помощью библиотеки *nanomsg*. Очереди построены поверх разных протоколов, например, со стороны клиента поверх протокола websocket. В качестве формата передаваемых данных используется бинарный формат *cbor*.

Уровень информации. На данном уровне определяются данные, с которыми работает система, способы их хранения, описание событий и их предварительная обработка. В модуле графического интерфейса на этапе инициализации в качестве данных выступают статические файлы, которые передаются от других модулей системы управления. К статическим файлам относятся:

- html файлы, описывающие структуру графического интерфейса;
- css файлы, описывающие внешний вид графического интерфейса;
- js файлы, описывающие логику работы графического интерфейса.

Для хранения данных модуль графического интерфейса использует базу данных UnQLite [26]. Данная система является NoSQL⁴ базой данных, основная особенность которой – отсутствие сервера. В отличие от таких известных NoSql движков как MongoDB⁵, Redis⁶, CouchDB⁷, UnQLite не требует установки или настройки, а также не запускает отдельного процесса для доступа к данным. Все данные хранятся в виде одного файла с сериализованными по стандарту JSON данными. В остальном, UnQLite является хранилищем пар «ключ-значение», с поддержкой курсоров и возможностью хранить данные как на диске, так и в оперативной

⁴NoSQL – подход к реализации хранилищ данных, где доступ к данным осуществляется без использования языка SQL.

⁵MongoDB – документо-ориентированная СУБД, использующая документы, основанные на формате JSON.

⁶Redis – сетевое хранилище типа «ключ-значение», осуществляющее хранение в оперативной памяти.

⁷CouchDB – документо-ориентированная СУБД, написанная на Erlang.

памяти. В дополнение к этому, UnQLite может работать и как хранилище документов и поддерживает транзакции.

После этапа инициализации модуль работает с данными, вводимыми пользователями, например, G-код программами, Gerber файлами и т. д.

Уровень функционала. На данном уровне определяется перечень функций, которые предлагаются для использования извне, и платформа для их предоставления.

Фактически сам по себе модуль графического интерфейса не обладает функционалом, доступным для других участников, за исключением удаленного доступа. Однако данный модуль является платформой, которую используют участники гетерогенной сети системы управления технологическим оборудованием для того, чтобы предоставить возможность воспользоваться своими функциями.

Учитывая вышесказанное, очевидно, что модуль графического интерфейса занимает заданную нишу в RAMI 4.0 и при этом следует определенной в RAMI 4.0 иерархии уровней. Таким образом, можно сделать вывод о том, что на базе данного модуля можно с успехом построить компонент Индустрии 4.0.

2.6 Выводы к второй главе

1. Пользовательский интерфейс технологического оборудования с ЧПУ имеет множество функций, в том числе связанных с визуализацией процесса обработки. Для их более эффективной реализации необходимо применять графический интерфейс, использующий все возможности веб-технологий.
2. В настоящее время существует значительное количество инструментов разработки пользовательского графического интерфейса. Чтобы идти в ногу со временем и удовлетворять требованиям четвертой

промышленной революции, необходимо использовать инструменты, поддерживающие современные веб-технологии.

3. Текущие подходы к реализации пользовательского графического интерфейса в модульных системах управления технологическим оборудованием влекут за собой ряд проблем. Структурно-модульный подход предлагает один из вариантов их решения.
4. Благодаря проведенному отображению модуля графического интерфейса на эталонную модель RAMI 4.0 было доказано, что модуль занимает вполне определенную нишу в RAMI 4.0 и при этом следует указанной в RAMI 4.0 иерархии уровней. Таким образом, можно сделать вывод о том, что на базе данного модуля можно с успехом построить компонент Индустрии 4.0. Данный факт полностью обосновывает применение предложенного подхода к разработке графического интерфейса.

Глава 3

Реализация модуля графического интерфейса

3.1 Описание реализованного модуля графического интерфейса

Реализованный модуль графического интерфейса состоит из двух частей: серверной и клиентской. Клиентская часть разбита на множество наборов виджетов, которые распределены по узлам гетерогенной сети системы управления технологическим оборудованием.

Расположение частей модуля пользовательского графического интерфейса на аппаратных ресурсах приведено на UML диаграмме развертывания (рис. 3.1). При этом в качестве примера рассматривается разрабатываемое модульное оборудование Adapteq [27] с установленной интеллектуальной лазерной головкой.

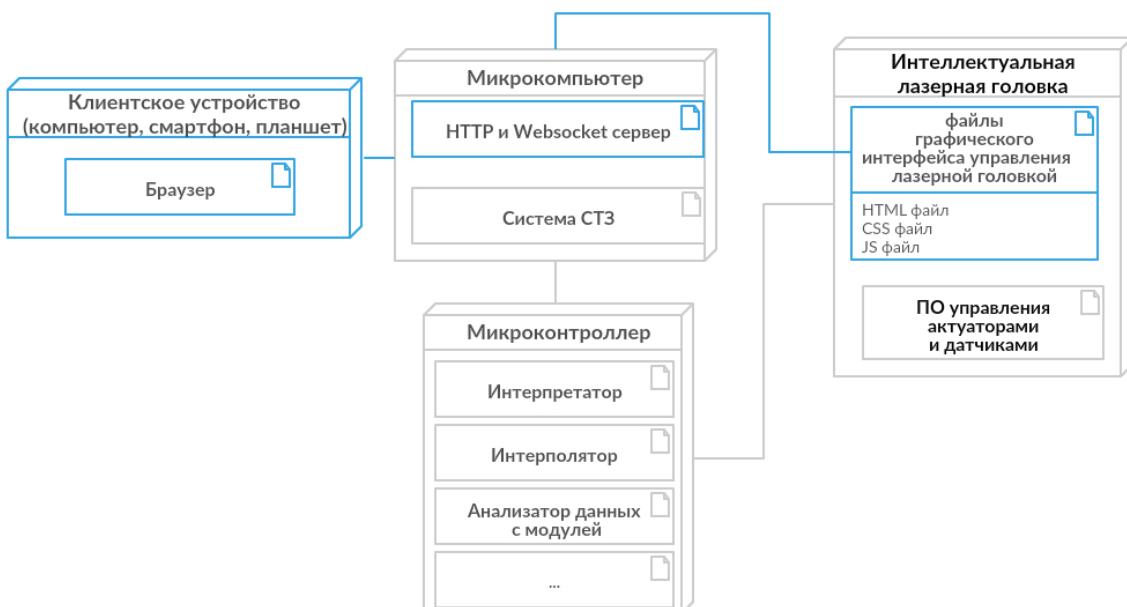


Рисунок 3.1 — UML диаграмма развертывания модуля графического интерфейса

Основой клиентской части служит оболочка, в которой реализованы окно авторизации пользователя, меню с наборами виджетов, упорядоченными по категориям, и редактируемое рабочее пространство. Данная оболочка по умолчанию предлагает заранее организованное рабочее пространство, однако оставляет пользователю возможность изменять его по своему желанию (например, удалять или добавлять виджеты, масштабировать их и т. д.).

Оболочка представляет собой html файл с соответствующими css файлами, где описан её внешний вид, и js файлы, где приведена логика отвечающая за работу меню, обработку жестов при сенсорном управлении и другие вещи важные для корректной работы графического интерфейса (рис. 3.2).

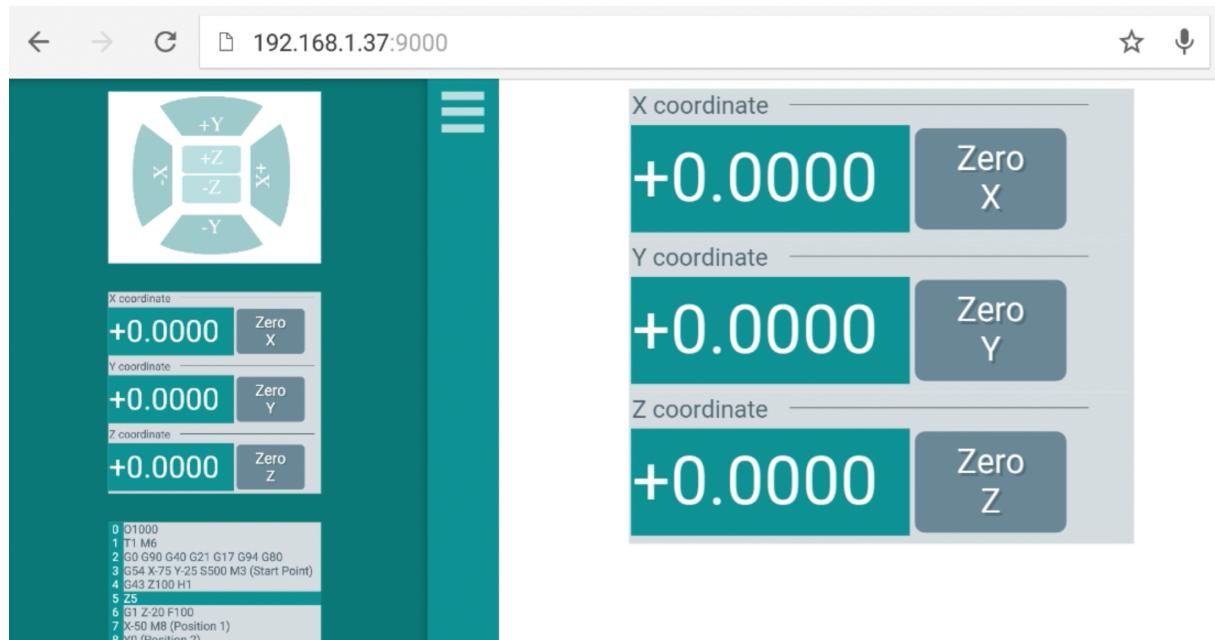


Рисунок 3.2 — Внешний вид оболочки

Виджеты встраиваются в клиентскую оболочку с использованием фреймов. Использование фреймов позволяет встраивать в базовый документ любые независимые фрагменты, при этом отображая их в заданной области определенного размера. Виджет, также как и оболочка, является html файлом с набором связанных с ним css и js файлов. Внутри js файлов кроме логики работы графической части содержатся обработчики,

позволяющие устанавливать соединение с сервером и обмениваться с ним данными по WebSocket (рис. 3.3).



Рисунок 3.3 — Внутренняя структура виджета

Серверная часть модуля графического интерфейса представлена HTTP и WebSocket сервером. Через HTTP-соединение пользователю передается оболочка графического интерфейса, а через WebSocket-соединение происходит асинхронное взаимодействие с виджетами. После авторизации пользователя обмен идет через безопасное HTTPS- и WSS-соединение с шифрованием. Сервер реализован на языке Go с использованием фреймворка Revel (рис. 3.4).

Сервер открывает для «прослушивания» набор веб-сокетов. Количество открытых сокетов соответствует количеству наборов виджетов с одной стороны, а с другой стороны открывает сокеты для взаимодействия в распределенной сети системы ЧПУ.

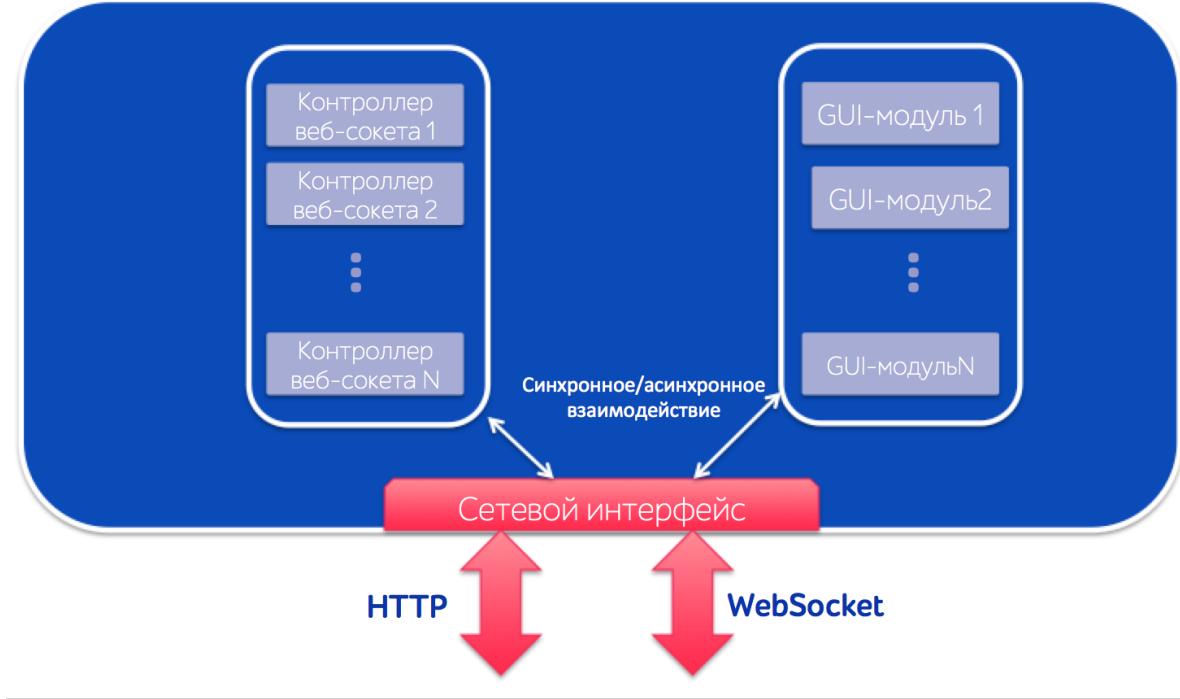


Рисунок 3.4 — Внутренняя структура сервера

3.2 Описание взаимодействия реализованного модуля графического интерфейса

Для взаимодействия серверной части модуля с клиентской частью, т. е. виджетами пользовательского графического интерфейса, используется протокол *WebSocket*. Данный протокол является полнодуплексным¹ и асинхронным. Это позволяет осуществлять взаимодействие клиента и системы ЧПУ в условиях близких к условиям реального времени, так называемое «мягкое реальное время». Более того, протокол Websocket имеет версию, поддерживающую шифрование WSS. На текущий момент все популярные браузеры полностью поддерживают этот протокол.

Для реализации взаимодействия используется *WebSocket API*, доступное в *JavaScript*. Для установки соединения с сервером применяется «рукопожатие» – процедура, при которой клиент и сервер посыпают друг другу запросы на соединение ([рис. 3.5](#)).

¹Означает, что передача и отправка могут осуществляться одновременно.

Однако для унификации протокола взаимодействия во всей системе управления технологическим оборудованием поверх протокола Websocket осуществляется взаимодействие на основе библиотеки *panotsg*. При этом используется тип взаимодействия «подписка-издатель». Для этого на клиентской части достаточно добавить параметр при вызове конструктора Websocket.

Разработанный модуль осуществляет взаимодействие, разделяя этот процесс на два этапа: этап инициализации и рабочий этап. На этапе инициализации осуществляется опрос модулей системы ЧПУ. В сеть посыпается широковещательный запрос, и те модули, которые могут на него ответить, посыпают на сервер модуля графического интерфейса наборы виджетов и метаданные (рис. 3.6). Во время же рабочего этапа осуществляется обмен полезными данными (командами, данными модулей, управляющими программами и т. д.).

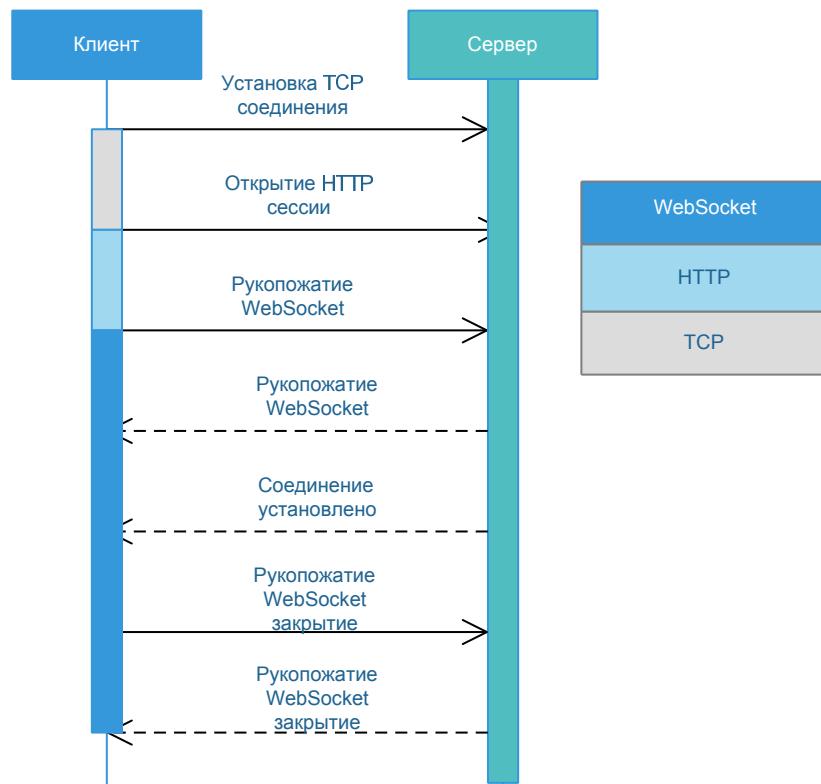


Рисунок 3.5 — Диаграмма последовательности рукопожатия WebSocket

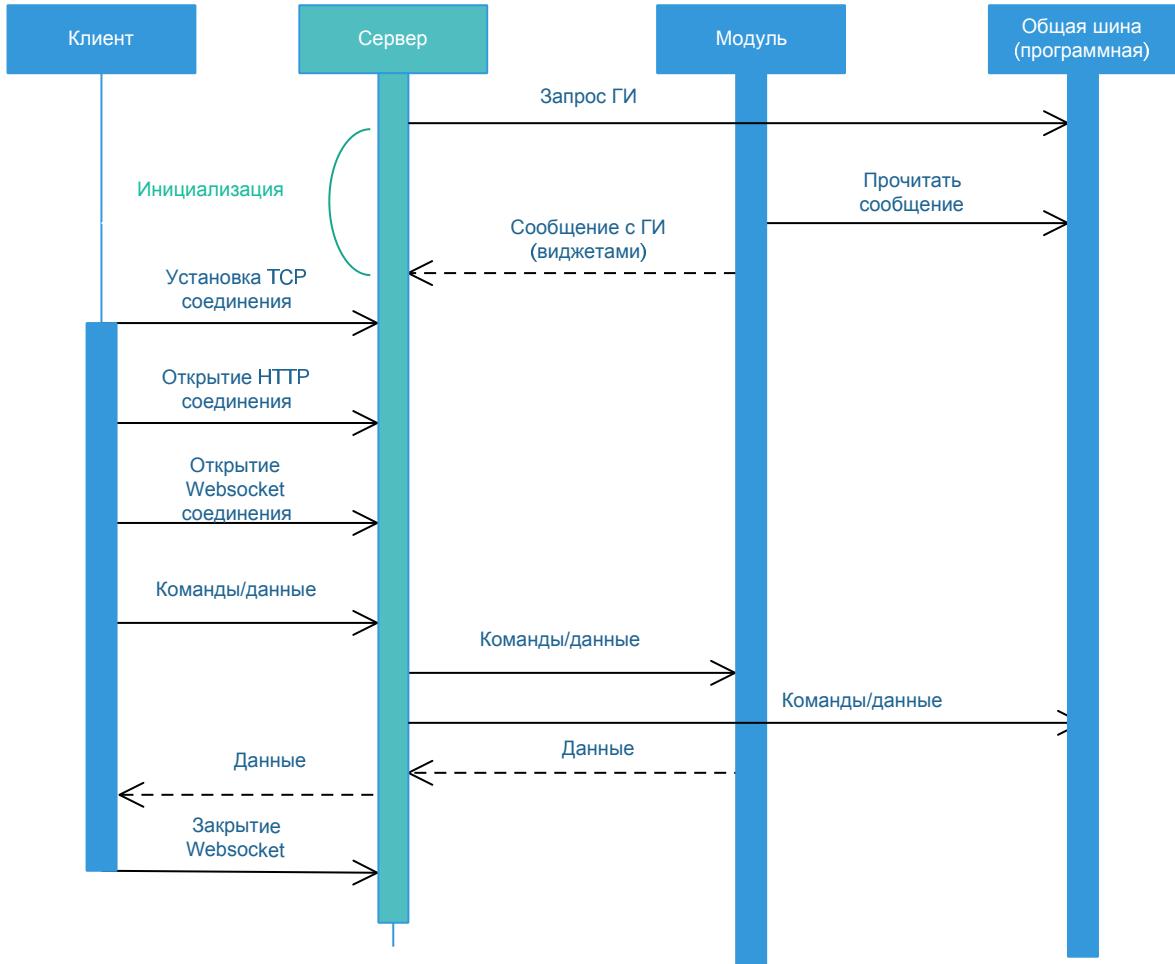


Рисунок 3.6 — Диаграмма последовательности модуля графического интерфейса

3.3 Состав технических средств

Серверная часть программного обеспечения была развернута на аппаратной платформе ODROID C2. Платформа ODROID C2 – это 64-битный одноплатный четырехпроцессорный микрокомпьютер с установленной операционной системой Ubuntu Mate. В составе системы управления ЧПУ данный модуль связывается с остальными модулями системы через Ethernet, UART и WiFi. Технические характеристики данного микрокомпьютера приведены в таблице Г.1.

Для обеспечения доступа к беспроводной сети, был использован WiFi-адаптер. Как видно из таблицы Г.1 микрокомпьютер не имеет встроенного WiFi-модуля, поэтому был использован внешний адаптер, подключаемый по USB. Его технические характеристики указаны в таблице 3.2.

Таблица 3.1 — Технические характеристики Odroid-C2.

Характеристика	Значение
Процессор	Amlogic ARM Cortex-A53(ARMv8) 1.5ГГц
Графический процессор	Mali-450 GPU
Память	2Gbyte DDR3 SDRAM
Ethernet	Gigabit Ethernet
USB	USB 2.0 4 порта
Слоты флеш-памяти	UHS-1 SDR50 MicroSD, eMMC5.0 HS400
USB OTG	USB 2.0 OTG 1 порт
Ввод/вывод	40pin GPIOs + 7pin I2S
Размер, мм	85×56×18
Вес, г	40

Таблица 3.2 — Технические характеристики Mediatek Ralink RT5370N2.

Характеристика	Значение
Интерфейс подключения	USB 2.0
Стандарт WiFi	IEEE 802.11 b/g/n
Частота	2.4 ГГц
Размер, мм	22.5×13.8×6.5
Вес, г	3.5

3.4 Выводы к третьей главе

- Поскольку важно, чтобы модуль графического интерфейса работал в режиме «мягкого реального времени», то для увеличения его производительности и скорости работы необходимо использовать компилируемый язык с продвинутой поддержкой асинхронности и многопоточности. Одним из таких языков программирования является язык Go.

2. При организации взаимодействия в целях обеспечения безопасности важно использовать протоколы, поддерживающие шифрование. Более того, следует стремиться к унификации протокола во всей системе управления.
3. Разработанный графический модуль не предъявляет высокие требования к аппаратным ресурсам. Поэтому для его размещения достаточно микрокомпьютера, либо обладающего встроенным WiFi-адаптером, либо имеющего интерфейс, через который возможно подсоединить внешний WiFi-адаптер.

ЗАКЛЮЧЕНИЕ

Проведенные исследования показали, что промышленность активно развивается в направлении повсеместного внедрения информационных и коммуникационных технологий. Более того, происходит интеллектуализация всех участников производства, что сопровождается увеличением их автономности и независимости. Изменяется парадигма управления как производством в целом, так и отдельно взятыми компонентами.

Технологическое оборудование – это неотъемлемый участник производственного процесса. Поэтому вопросы организации ЧПУ системы и её компонентов в соответствии с современными тенденциями развития производства являются актуальными.

Человеко-машинный интерфейс – это один из основных элементов системы ЧПУ. Предложенный структурно-модульный подход к построению пользовательского графического интерфейса, как узла распределенной сети системы ЧПУ, позволяет осуществлять управление модульным технологическим оборудованием.

В ходе выполнения работы были достигнуты следующие результаты:

- Было проведено исследование предметной области и современных разработок в области создания систем ЧПУ.
- Было проведено описание структурно-модульного подхода к разработке пользовательских графических интерфейсов.
- Было обосновано применение структурно-модульного подхода с помощью отображения его на эталонную архитектуру RAMI 4.0.
- Была проведена классификация и обзор инструментов создания пользовательского графического интерфейса.
- Был разработан модуль пользовательского графического интер-

файса систему управления технологическим оборудованием, в частности его серверная часть, включающая HTTP- и WebSocket-сервер, и клиентская часть, включающая библиотеку базовых виджетов.

ПЕРЕЧЕНЬ РИСУНКОВ

2.1	Стойка FANUC Series 0i-TD	22
2.2	Стойка KSE-TOUCH CNC	22
2.3	Агрегатор	33
2.4	Структура системы с иерархической организацией	33
2.5	Составляющие модульного оборудование с ЧПУ	34
2.6	Организация компонента графического интерфейса	35
2.7	Архитектурная модель RAMI 4.0	39
3.1	UML диаграмма развертывания модуля графического интерфейса	44
3.2	Внешний вид оболочки	45
3.3	Внутренняя структура виджета	46
3.4	Внутренняя структура сервера	47
3.5	Диаграмма последовательности рукопожатия Websocket .	48
3.6	Диаграмма последовательности модуля графического интерфейса	49

ПЕРЕЧЕНЬ ТАБЛИЦ

3.1	Технические характеристики Odroid-C2.	50
3.2	Технические характеристики Mediatek Ralink RT5370N2.	50

СПИСОК ИСПОЛЬЗУЕМЫХ СОКРАЩЕНИЙ

API — Application programming interface

BPEL — Business process execution language

CSS — Cascading style sheets

ERP — Enterprise resource planning

FPGA — Field-programmable gate array

HTML — HyperText markup language

IIRA — Industrial internet reference architecture

IP — Internet protocol

JS — JavaScript

JSON — JavaScript object Notation

JSX — JavaScript XML

HTTP — HyperText transfer protocol

MDI — Manual data input

MES — Manufacturing execution system

MSA — Microservice architecture

OSI — Open systems interconnection

RAMI — Reference architectural model Industrie 4.0

SCADA — Supervisory control and data acquisition

SOA — Service-oriented architecture

SOAP — Simple object access protocol

TCP — Transmission control protocol

UART — Universal asynchronous receiver-transmitter

UML — Unified modeling language

XML — Extensible markup language

XMP-RPC — Extensible markup language remote procedure call

WSDL — Web services description language

WSS — WebSocket secure

АСУ ТП — Автоматизированная система управления технологическим процессом

ГИ — Графический интерфейс

ППВМ — Программируемая пользователем вентильная матрица

ЧПУ — Числовое программное управление

СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Jose C., Pedro F. Wireless Sensors in Industrial Time-Critical Environments. Springer International Publishing, 2014. ISBN: 978-3-319-02888-0.
2. Winnum Platform [Электронный ресурс]. URL: <http://www.winnum.ru> (дата обращения: 12.04.2017).
3. User Manual for your ODROID-C2 [Электронный ресурс]. URL: <http://odroid.com/dokuwiki/doku.php?id=en:odroid-c2> (дата обращения: 16.04.2017).
4. Morales-Velazquez L., de Jesus Romero-Troncoso R., Osornio-Rios R. A. et al. Open-architecture system based on a reconfigurable hardware-software multi-agent platform for CNC machines // Journal of Systems Architecture. 2010. Vol. 56, no. 9. P. 407 – 418.
5. Grigoriev S. N., Martinov G. M. Research and Development of a Cross-platform CNC Kernel for Multi-axis Machine Tool // Procedia CIRP. 2014. Vol. 14. P. 517–522. 6th CIRP International Conference on High Performance Cutting, HPC2014.
6. IEC 61158-1. Industrial communication networks - Fieldbus specifications - Part 1: Overview and guidance for the IEC 61158 and IEC 61784 series: Standard: International electrotechnical commission, 2014.
7. IEC 61800-7-1:2015. Adjustable speed electrical power drive systems - Part 7-1: Generic interface and use of profiles for power drive systems - Interface definition: Standard: International electrotechnical commission, 2015.
8. IEC 61784-1:2014. Industrial communication networks - Profiles - Part 1: Fieldbus profiles: Standard: International electrotechnical commission, 2014.
9. ISO 15745-4:2003. Industrial automation systems and integration – Open systems application integration framework – Part 4: Reference description

- for Ethernet-based control systems: Standard: International Organization for Standardization, 2003.
10. ISO 11898-1:2015. Road vehicles – Controller area network (CAN) – Part 1: Data link layer and physical signalling: Standard: International Organization for Standardization, 2015.
 11. Modicon Modbus Protocol Reference Guide. URL: <http://www.schneider-electric.com/cn/en/download/document/Modbus+protocol+Reference+Guide> (дата обращения: 16.04.2017).
 12. Bin L., Yun-fei Z., Xiao-qи T. A research on open CNC system based on architecture/component software reuse technology // Computers in Industry. 2004. Vol. 55, no. 1. P. 73 – 85.
 13. IEC 62541-100:2015 OPC Unified Architecture - Part 100: Device Interface: Standard: International electrotechnical commission, 2015.
 14. IEC 61987-11:2016 Industrial-process measurement and control - Data structures and elements in process equipment catalogues - Part 11: List of properties (LOPs) of measuring equipment for electronic data exchange - Generic structures: Standard: International electrotechnical commission, 2016.
 15. DIN SPEC 40912:2014-11 Core models - Specification and Examples: Standard: German Institute for Standardization, 2014.
 16. DIN SPEC 91345:2016-04 Reference Architecture Model Industrie 4.0 (RAMI4.0): Standard: German Institute for Standardization, 2016.
 17. Lin S.-W., Miller B., Durand J. et al. The Industrial Internet of Things Volume G1: Reference Architecture: Tech. rep.: Industrial Internet Consortium, 2017.
 18. Laskey K. B., Laskey K. Service oriented architecture // Wiley Interdisciplinary Reviews: Computational Statistics. 2009. Vol. 1, no. 1. P. 101–105.
 19. SOA Manifesto [Электронный ресурс]. URL: www.soa-manifesto.org

(дата обращения: 23.04.2017).

20. Microservices. A definition of this new architectural term. [Электронный ресурс]. URL: <https://martinfowler.com/articles/microservices.html> (дата обращения: 23.04.2017).
21. Suh S.-H., Kang S.-K., Chung D.-H., Stroud I. Theory and Design of CNC Systems. Springer-Verlag, 2008. 455 p.
22. Синяков А. И. Анализ модульного подхода и его применение в различных языках программирования // Методы и инструменты конструирования и оптимизации программ. 2005. Т. 1, № 1. С. 197–228.
23. Разработка ПО. Этапы разработки программного обеспечения. [Электронный ресурс]. URL: http://ab-solut.net/ru/articles/etapi_po/ (дата обращения: 30.04.2017).
24. Структурный подход к проектированию ИС. [Электронный ресурс]. URL: http://citforum.ru/database/case/glava2_1.shtml (дата обращения: 30.04.2017).
25. Stevens W. R. TCP/IP Illustrated (Vol. 1): The Protocols. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1993. ISBN: 0-201-63346-9.
26. An Embeddable NoSQL Database Engine. [Электронный ресурс]. URL: <https://unqlite.org/index.html> (дата обращения: 25.04.2017).
27. Афанасьев М. Я., Грибовский А. А. Концепция адаптивной платформы технологического оборудования // Известия высших учебных заведений. Приборостроение. 2015. Т. 58, № 4. С. 268–272.

ПРИЛОЖЕНИЕ А

Программная реализация обработчиков основных запросов на сервер

Листинг А.1 — Реализация обработчиков основных запросов на сервер

```
1 package controllers
2
3 import (
4     "servergui/app"
5     "servergui/app/streamcam"
6     "servergui/app/webterm"
7     "strconv"
8
9     "github.com/revel/revel"
10    "golang.org/x/net/websocket"
11 )
12
13 type WebSockApp struct {
14     *revel.Controller
15 }
16
17 func (c WebSockApp) Index() revel.Result {
18     revel.INFO.Printf("%s", "GET request received")
19     host := app.AddrHTTP
20     port := app.PortHTTP
21     return c.Render(host, port)
22 }
23
24 func (c WebSockApp) StreamWidget() revel.Result {
25     revel.INFO.Printf("%s", "GET request received")
26     host := app.AddrHTTP
27     port := app.PortHTTP
```

```

28     return c.Render(host, port)
29 }
30
31 func (c WebSockApp) AxisMovementsWidget() revel.Result {
32     revel.INFO.Printf("%s", "GET request received")
33     host := app.AddrHTTP
34     port := app.PortHTTP
35     return c.Render(host, port)
36 }
37
38 func (c WebSockApp) CoordInfoWidget() revel.Result {
39     revel.INFO.Printf("%s", "GET request received")
40     host := app.AddrHTTP
41     port := app.PortHTTP
42     return c.Render(host, port)
43 }
44 func (c WebSockApp) GCodeWidget() revel.Result {
45     revel.INFO.Printf("%s", "GET request received")
46     host := app.AddrHTTP
47     port := app.PortHTTP
48     return c.Render(host, port)
49 }
50
51 func (c WebSockApp) WSHandler(usr string,
52                                 ws *websocket.Conn)
53                                 revel.Result {
54     defer ws.Close()
55     revel.INFO.Printf("%s", "WS request received")
56     newmessages := make(chan string)
57     quit := make(chan struct{})
58     go streamcam.StreamVideo(ws, quit, app.GetCapture())
59     go func() {
60         var msg string
61         for {
62             err := websocket.Message.Receive(ws, &msg)

```

```
63         if err != nil {
64             close(newmessages)
65             return
66         }
67         newmessages <- msg
68     }
69 }()
70 for {
71     select {
72     case msg := <-newmessages:
73         if msg == "" {
74             close(quit)
75             revel.INFO.Printf("%s", "Closed")
76             return nil
77         }
78         revel.INFO.Printf("WS message %s", msg)
79     }
80 }
81 }
```

ПРИЛОЖЕНИЕ Б

HTML реализация графической оболочки

Листинг Б.1 — Реализация обработчиков основных запросов на сервер

```
1 <html>
2     <head>
3         <meta charset="utf-8">
4         <title>Websocket server</title>
5     </head>
6     <body>
7         <link rel="stylesheet"
8             href="/public/css/video-stream.css"
9             type="text/css">
10        </link>
11        <script src="/public/js/main-init.js"
12            type="text/javascript">
13        </script>
14        <script src="/public/js/vendors/hammer.min.js">
15        </script>
16        <div id="menu">
17            <div id="menu-btn" onclick="menuShow()">
18            </div>
19            <div id="content" style="overflow-y: scroll;">
20                <div class="menu-item"
21                    onclick="addAxisMovementsWidget()">
22                    
25                </div>
26                <div class="menu-item"
27                    onclick="addCoordInfoWidget()">
28                    
```

```
30                     alt= "stream-widget"
31                         width = "150">
32             </div>
33             <div class= "menu-item"
34                 onclick= "addGCodeWidget () ">
35                 <img id = "onfo-widget-icon"
36                     src= "/public/img/gcode.png"
37                     alt= "stream-widget" width = "150">
38             </div>
39             <div class= "menu-item"
40                 onclick= "addStreamWidget () ">
41                 <img id = "stream-widget-icon"
42                     src= "/public/img/stream.png"
43                     alt= "stream-widget"
44                     width = "150">
45             </div>
46         </div>
47     </div>
48     <div id= "workspace">
49         </div>
50     </body>
51 </html>
```

ПРИЛОЖЕНИЕ В

Графические формы

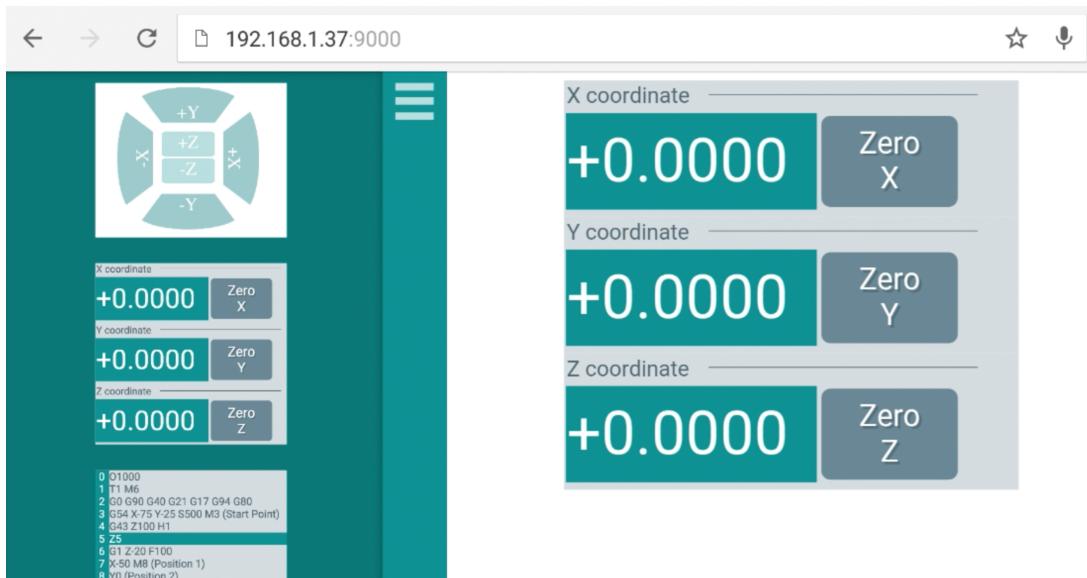


Рисунок В.1 — Общий вид графической оболочки

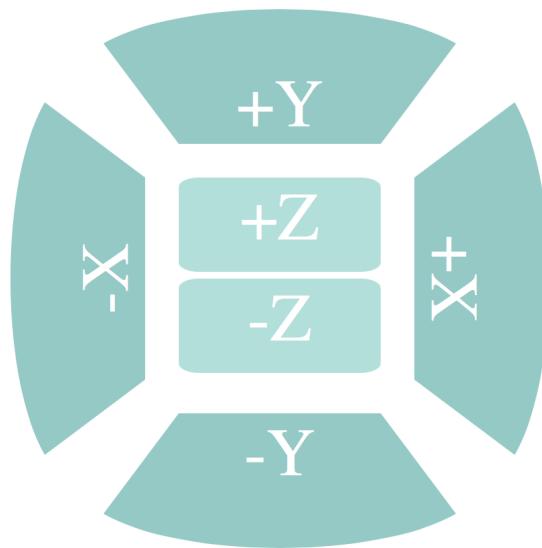


Рисунок В.2 — Виджет управления движения по осям

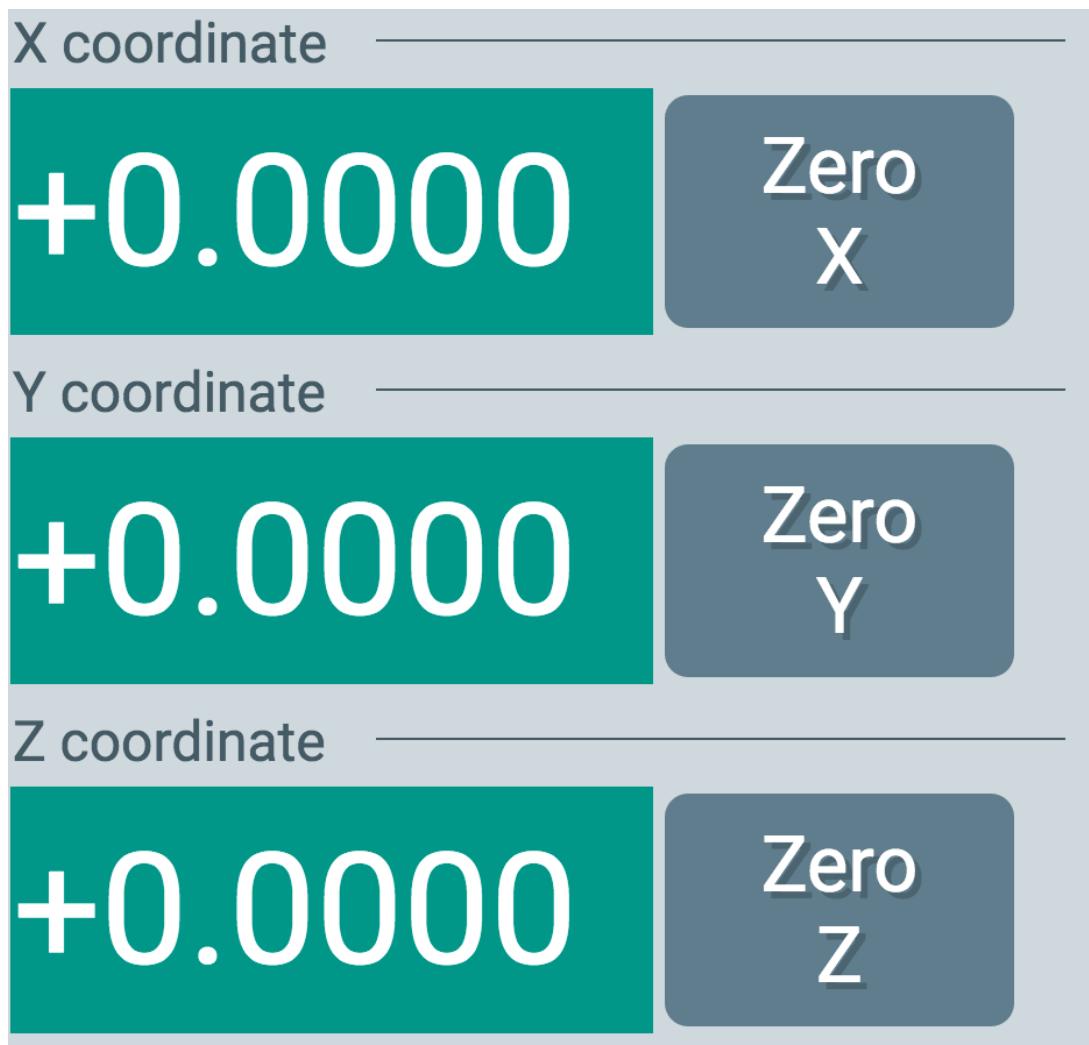


Рисунок В.3 — Виджет отображения координат и установки нуля

0	01000
1	T1 M6
2	G0 G90 G40 G21 G17 G94 G80
3	G54 X-75 Y-25 S500 M3 (Start Point)
4	G43 Z100 H1
5	Z5
6	G1 Z-20 F100
7	X-50 M8 (Position 1)
8	Y0 (Position 2)
9	X0 Y50 (Position 3)

Рисунок В.4 — Виджет отображения управляющей программы

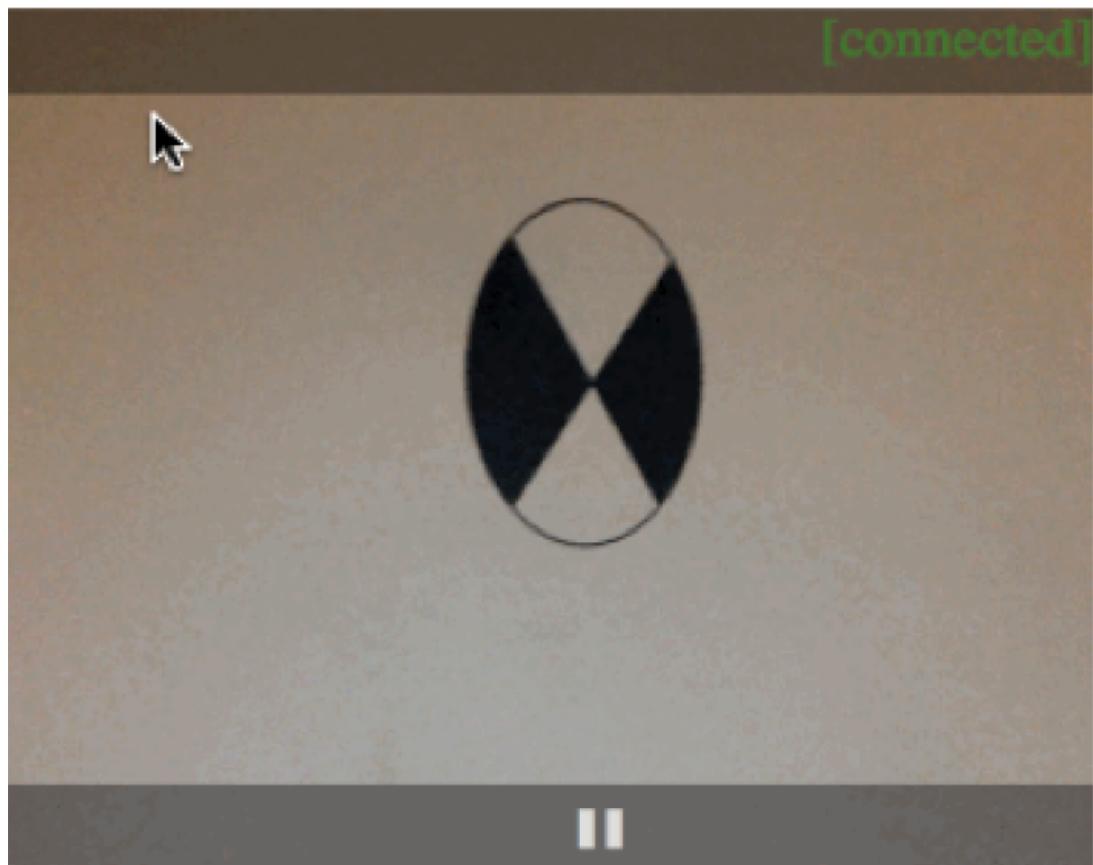


Рисунок В.5 — Виджет отображения видео-потока

```
.-----.
|  _ _ _ _ |    ( )    | _ _ _ _ |    \ \ / /
!  | | - - - - - - - - | | - - - - - - - - / / !
:  | | ' _ / _ \ | / _ \ \ / | / _ \ \ / ' _ \ / : .
.  | | | | ( ) | | _ / ( ) | | ( ) | | | | | .
:  | _ | | \ _ , _ | | \ _ , _ | | \ _ , _ | | \ _ , _ | : .
!  _ / |
|   ver. 0.0 | _ /      Trajectory CNC      |
`-----'

Type "help" to view a list of available commands.

Trj > help
Press ENTER to execute the previous command again.
Type "man <command>" to get the detailed description of the command.

List of available commands:

  help
  man
  start
  led
  task-stats
  echo-3-parameters
  echo-parameters
```

Рисунок В.6 — Виджет терминала

ПРИЛОЖЕНИЕ Г

Микрокомпьютер ODROID-C2

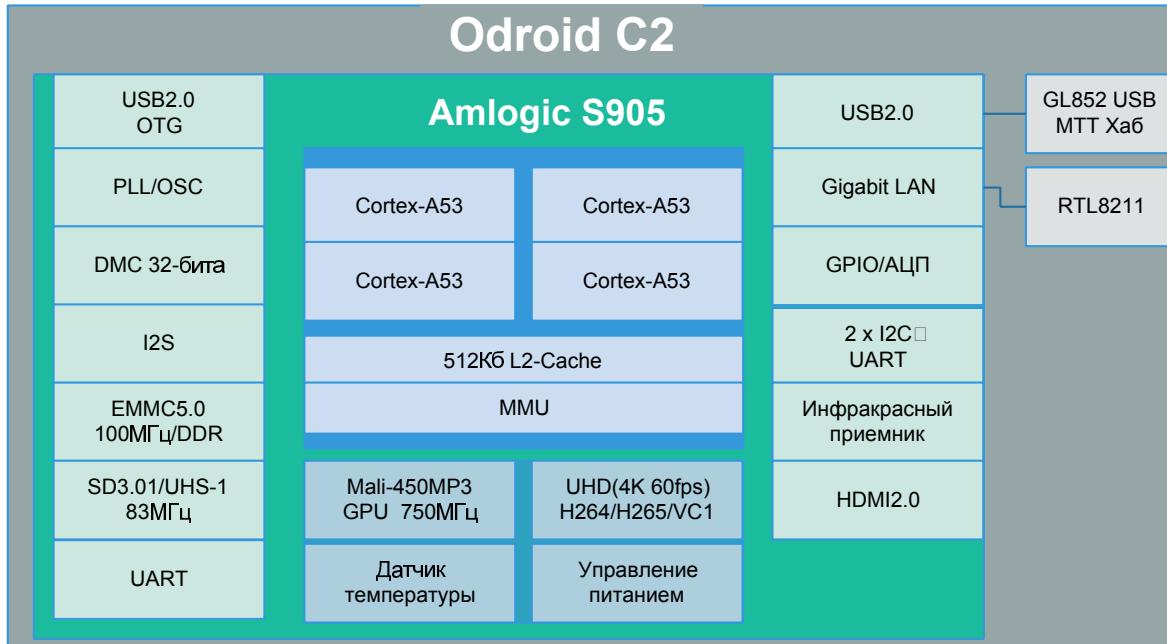


Рисунок Г.1 — Блок-схема ODROID-C2

Таблица Г.1 — Технические характеристики Odroid-C2

Характеристика	Значение
Процессор	Amlogic ARM Cortex-A53(ARMv8) 1.5ГГц
Графический процессор	Mali-450 GPU
Память	2Gbyte DDR3 SDRAM
Ethernet	Gigabit Ethernet
USB	USB 2.0 4 порта
Слоты флеш-памяти	UHS-1 SDR50 MicroSD, eMMC5.0 HS400
USB OTG	USB 2.0 OTG 1 порт
Ввод/вывод	40pin GPIOs + 7pin I2S
Размер, мм	85×56×18
Вес, г	40

ПРИЛОЖЕНИЕ Д

Общий вид координатной платформы

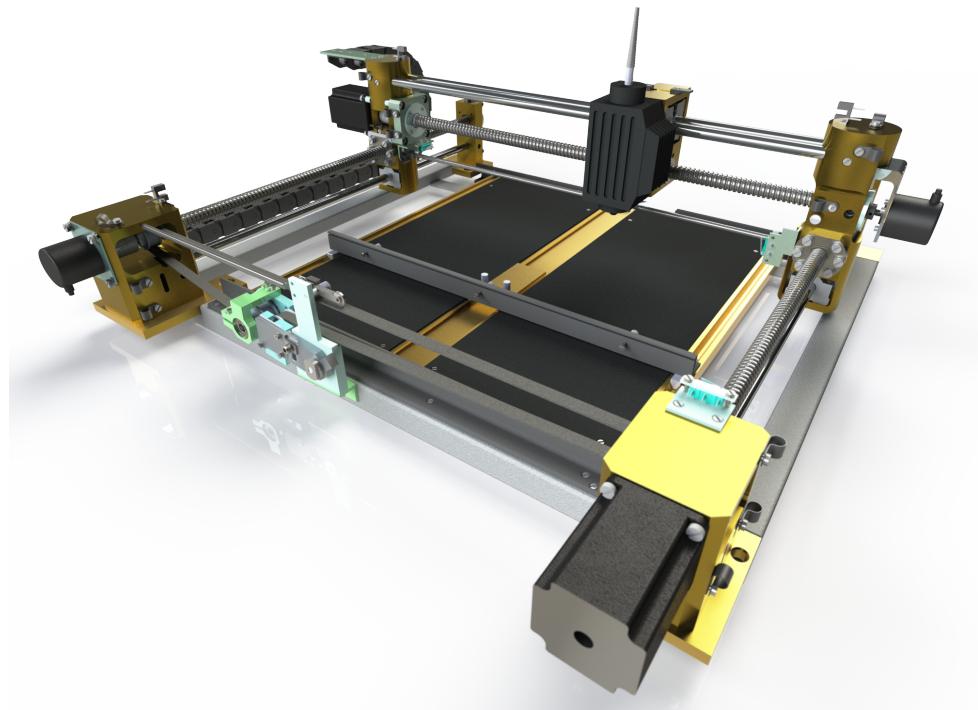


Рисунок Д.1 — 3D-модель координатной платформы



Рисунок Д.2 — Фотография координатной платформы