

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ

«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Параллельных вычислительных технологий
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.

(подпись, инициалы, фамилия)

«__» _____ 2020 г.

МАГИСТЕРСКАЯ ДИССЕРТАЦИЯ
по направлению высшего образования

01.04.02 Прикладная математика и информатика
(код и наименование направления подготовки магистра)

Факультет прикладной математики и информатики
(факультет)

Мошкина Алёна Дмитриевна
(фамилия, имя, отчество студента – автора работы)

Разработка программной системы для реализации пользовательских интерфейсов
(полное название темы магистерской диссертации)
вычислительных приложений, работающих под управлением HPC Community Cloud
на высокопроизводительных вычислительных системах

**Руководитель
от НГТУ**

Малышкин В. Э.
(фамилия, имя, отчество)

д.т.н, профессор
(ученая степень, ученое звание)

(подпись, дата)

**Автор выпускной
квалификационной
работы**

Мошкина А.Д.
(фамилия, имя, отчество)

ФПМИ, ПММ-82
(факультет, группа)

(подпись, дата)

Новосибирск 2020

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра Параллельных вычислительных технологий
(полное название кафедры)

УТВЕРЖДАЮ

Зав. кафедрой Малышкин В.Э.

(подпись, инициалы, фамилия)

«__» _____ 2020 г.

**ЗАДАНИЕ
НА МАГИСТЕРСКУЮ ДИССЕРТАЦИЮ**

студенту Мошкиной Алёне Дмитриевне

(фамилия, имя, отчество)

факультета Прикладной математики и информатики

(полное название факультета)

Направление подготовки 01.04.02 Прикладная математика и информатика

(код и наименование направления подготовки магистра)

Магистерская программа Математическое моделирование детерминированных и
стохастических процессов

(наименование магистерской программы)

Тема Разработка программной системы для реализации пользовательских

(полное название темы)

интерфейсов вычислительных приложений, работающих под управлением

HPC Community Cloud на высокопроизводительных вычислительных системах

Цели работы разработка программной системы, предоставляющей пользователям

инструментарий для реализации пользовательских интерфейсов вычислительных

приложений, работающих на высокопроизводительных вычислительных

системах (BBC) под управлением сервиса HPC Community Cloud (HPC2C)

Задание согласовано и принято к исполнению.

**Руководитель
от НГТУ**

Студент

Мальшикин В. Э.

Мошкина А.Д.

д.т.н, профессор

ФПМИ, ПММ-82

(ученая степень, ученое звание)

(факультет, группа)

(подпись, дата)

(подпись, дата)

Тема утверждена приказом по НГТУ № 5110/2 от « 18 » октября 2018 г.
изменена приказом по НГТУ № 1242/2 от « 02 » марта 2020 г.

Диссертация сдана в ГЭК № _____, тема сверена с данными приказа

(подпись секретаря государственной экзаменационной комиссии по защите ВКР, дата)

(фамилия, имя, отчество секретаря государственной экзаменационной комиссии по защите ВКР)

АННОТАЦИЯ

Работа состоит из 96 страниц, 3 частей, 40 рисунков, 1 таблицы, списка литературы из 32 источников, 6 приложений.

ВЫСОКОПРОИЗВОДИТЕЛЬНЫЕ ВЫЧИСЛИТЕЛЬНЫЕ СИСТЕМЫ, ПОЛЬЗОВАТЕЛЬСКИЕ ИНТЕРФЕЙСЫ, GUI, ГЕНЕРАЦИЯ ИНТЕРФЕЙСОВ, УПРАВЛЕНИЕ ВЫЧИСЛЕНИЯМИ, УНИФИЦИРОВАННЫЙ ДОСТУП, ВЫЧИСЛИТЕЛЬНЫЕ СЦЕНАРИИ, ОПИСАНИЕ ДАННЫХ.

Объектом исследования является генерация пользовательских интерфейсов для приложений, работающих на высокопроизводительных вычислительных системах (BBC).

Цель работы – разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах под управлением HPC Community Cloud (HPC2C).

В процессе работы изучались способы взаимодействия с высокопроизводительными вычислительными системами, формализации знаний о предметных областях, а также примеры систем для генерации интерфейсов, разрабатывался программный инструментарий, позволяющий генерировать интерфейсы приложений по их формальному описанию.

В результате предложен пользовательский инструментарий для генерации интерфейсов, позволяющий как воспользоваться готовыми модулями (веб-интерфейс, мобильный интерфейс), так и сгенерировать специфический для конкретного приложения интерфейс по предоставленному описанию данных, который становится доступен в рамках веб-приложения HPC2C или, обращаясь к API HPC2C, реализовать его как самостоятельное приложение.

Степень внедрения – разработанный инструментарий планируется применять в дальнейших исследованиях по теме работы на кафедре Параллельных вычислительных технологий НГТУ.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ	7
1. ОБЗОР И ПОСТАНОВКА ЗАДАЧИ	12
1.1. Обзор способов взаимодействия пользователей и/или интерфейсных систем с высокопроизводительными вычислительными системами	13
1.2. Формальное представление знаний	18
1.3. Исследование текущей версии HPC Community Cloud	23
1.4. Обзор систем для создания workflow и генерации интерфейсов	32
1.5. Формулирование требований к системе	35
1.5.1. Описание основных требований к системе	35
1.5.2. Варианты взаимодействия пользователя с системой	36
1.5.3. Постановка задачи	39
2. ПРОЕКТ	41
2.1. Описание инструментария для разработки интерфейсов	41
2.2. Базовая функциональность системы	43
2.2.1 Описание интерфейсной системы Aka	43
2.2.2 Проект мобильного приложения HPC Community Cloud	48
2.2.3 Описание профилировщика под управлением HPC Community Cloud	51
2.3. Инструменты для генерации специальных интерфейсов под конкретное вычислительное приложение	53
2.3.1 Описание конструктора типов и их спецификации	53
2.3.2. Классификация типов данных	54
2.3.3 Визуализация типов данных	58
2.3.4. Валидация объектов данных	59
2.3.5. Управление данными	61
2.4. Usability	62
2.5. Визуализация результатов вычислительного приложения	64
3. РЕАЛИЗАЦИЯ СИСТЕМЫ	67
3.1. Выбор средств для реализации программного комплекса	67

3.2. Описание модельного приложения	67
3.3. Демонстрация работы модельного приложения внутри системы НРС2С	69
3.3.1. Общий вид разрабатываемого вычислительного сценария	69
3.3.2. Проектирование вычислительного сценария для получения начального состояния клеточного автомата	71
3.3.3. Визуализация входных данных для моделирования клеточного автомата	75
3.4. Демонстрация работы мобильного приложения под управлением НРС2С	79
3.4.1. Основная функциональность приложения	79
3.4.2. Профилировщик под управлением НРС2С	81
ЗАКЛЮЧЕНИЕ	83
СПИСОК ЛИТЕРАТУРЫ	85
Приложение А	89
Приложение Б	90
Приложение В	91
Приложение Г	92
Приложение Д	93
Приложение Е	94

ВВЕДЕНИЕ

Многие численные задачи характеризуются большим объёмом данных и вычислений и не всякая вычислительная система за приемлемое время способна выдать результат для таких задач. Для таких крупномасштабных вычислений возникает необходимость использовать высокопроизводительные вычислительные системы (ВВС), следовательно, возникает задача разработки программ для таких систем и необходимость организации работы пользователя с системами такого типа. Организация пользовательского взаимодействия с ВВС включает в себя следующие основные операции: передача данных на ВВС, запуск вычислительных задач, что, как правило, означает постановку в очередь некоторой системы управления прохождением задач (СУПЗ), отслеживание статуса выполнения задачи, извлечение результатов задачи и последующий анализ.

Большинству прикладных специалистов сложно пользоваться высокопроизводительными вычислительными системами, так как существует множество различных видов таких систем и для успешной и быстрой работы с каждым таким типом вычислительной системы, необходимо предварительно изучить соответствующую документацию по конкретному типу вычислительной системы.

Также довольно трудоёмко работать с ВВС при использовании нескольких вычислительных систем для работы, как, например, делают пользователи из институтов Сибирского отделения Российской академии наук (СО РАН). В своей работе они применяют вычислительные кластеры Сибирского Суперкомпьютерного Центра (ССЦ), кластеры НГУ, МГУ, Межведомственного Суперкомпьютерного Центра РАН (МСЦ РАН), вычислительные машины, установленные в отдельных институтах и так далее.

Следовательно, возникает задача унификации доступа к ресурсам высокопроизводительных вычислительных систем.

Помимо проблем, связанных с выполнением основных операций по управлению задачами на BBC, существует проблема низкого уровня интерфейсов самих приложений. На практике не наблюдается, чтобы пользователям предоставлялись высокоуровневые интерфейсы доступа к вычислительным приложениям, работающим на высокопроизводительных вычислительных системах. Существуют пакеты, например, MATLAB, OpenFOAM, ANSYS Fluent и другие, направленные на конечного пользователя, где пользователь буквально рисует область, к примеру, сетку или пакеты, позволяющие декларативно задавать уравнения и получать их решения. Используя такие пакеты можно решать сложные вычислительные задачи на BBC, однако решение можно получить только в рамках конкретного пакета. При этом не все задачи могут быть решены с использованием таких частных пакетов, поэтому часто пользователи BBC разрабатывают собственные вычислительные программы, и отдельной трудностью для пользователей является разработка интерфейсов к таким программам. На разработку интерфейсов зачастую не хватает ресурсов. Одной из причин существования этой проблемы является отсутствие системного инструментария для разработки высокоуровневых интерфейсов и, в частности, отсутствие инструментария для создания визуализаторов, позволяющих визуализировать входные и выходные данные вычислительных экспериментов, а также готовых визуализаторов для результатов приложений, выполняющихся на BBC.

Если вычислительный эксперимент состоит из нескольких запусков, возможно различных программ, связанных указанным порядком выполнения, то помимо разработки интерфейсов к каждой программе на пользователя возлагается задача разработки скриптов управления запуском программ, сбора результатов, разработке визуализатора результатов, а также, если существует необходимость выполнения эксперимента на нескольких вычислительных системах для сравнения скорости выполнения, то и разработка модулей запуска эксперимента на интересующих системах. Если разработчику требуется поддерживать несколько подобных сложных вычислительных экспериментов, то разра-

ботка интерфейсов для таких приложений становится довольно объёмной и трудоёмкой.

В данный момент достаточно популярен клиент-серверный подход к разработке графических интерфейсов пользователя, для краткости GUI, причём проектируется тонкий клиент с минимально необходимым набором операций, а вся основная логика взаимодействия различных компонентов внутри GUI и самого GUI с другими системами располагается в серверной части. Также отдельно стоит выделить подход, при котором серверная часть развёртывается на некотором доступном в сети Интернет сервере, что позволяет использовать GUI в сети или же перенести на мобильные платформы.

Разработка интерфейсов такого типа для вычислительных экспериментов довольно трудоёмкий процесс, требующий помимо инструментов также знания в области проектирования и разработки GUI, а также клиент-серверного взаимодействия. Следовательно, возникает необходимость в системе, которая по описанию GUI на специально разработанном для неё языке будет автоматически генерировать интерфейс, необходимый пользователю данной системы в рамках мобильного-, web- или десктопного приложений и при необходимости визуализировать результаты работы вычислительного эксперимента одним или несколькими способами, доступными пользователю для выбора.

Кроме того, в зависимости от среды, в которой пользователь интерфейса работает, ему необходим соответствующий среде интерфейс, что тоже является трудоёмкой задачей, так как разработка интерфейса вычислительного эксперимента для как минимум трёх сред: браузерной, мобильной и десктопной предполагает наличие необходимых знаний в разработки GUI в каждой из сред, что увеличивает количество человекочасов, требуемых на разработку, кроме того, такая статья расходов требует соответствующего финансирования.

Целью данной работы является разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах под управлением

HPC Community Cloud (HPC2C – система для унификации доступа пользователей к ресурсам ВВС различного типа, проект ИВМиМГ СО РАН). Согласно рассмотренному выше такая программная система должна предоставить пользователям инструментарий для реализации интерфейсов вычислительных приложений и способов визуализации результатов работы приложений, выполняющихся на HPC2C, причём программная среда HPC2C будет предоставлять пользователям унифицированный пользовательский интерфейс для основных операций управления задачами и отправкой последних на выполнение на ВВС.

Подытожив всё вышеперечисленное, можно сделать вывод о том, что **актуальность работы** обусловлена следующими пунктами:

- отсутствием некоторого удобного сервиса, предоставляющего унифицированный доступ к ВВС для запуска задач;
- отсутствием удобных для использования интерфейсов к задачам, работающим удалённо на ВВС;
- отсутствием систем, позволяющих управлять сложными, многомодульными приложениями;
- отсутствием инструментария для разработки интерфейсов и для визуализации результатов запуска задач, работающих удалённо на ВВС;
- отсутствием интерфейсов вычислительных приложений, согласно среде, в которой в данный момент находится пользователь.

Предлагаемый подход к решению проблемы разработки интерфейсов приложений, выполняющихся на ВВС, подразумевающий использование сервиса HPC Community Cloud, системы для управления сложными приложениями на основе вычислительных моделей и средств генерации пользовательских интерфейсов по формальным спецификациям, а также соответствующий разработанный программный инструментарий обладают **научно-технической новизной**.

Практической значимостью обладает предлагаемый программный инструментарий для генерации пользовательских интерфейсов в совокупности, а

также его отдельные компоненты. Инструментарий включает в себя готовые интерфейсные модули, предоставляющие доступ к HPC Community Cloud в браузере и мобильном телефоне, разработанные вычислительные модели для управления вычислениями в рамках HPC Community Cloud и прикладная вычислительная модель, формализующая сценарии проведения вычислительных экспериментов в клеточно-автоматной модели FHP-MP, инструменты для генерации элементов интерфейса произвольных приложений на основе формального описания типов входных/выходных данных.

1. ОБЗОР И ПОСТАНОВКА ЗАДАЧИ

Проблема отсутствия удобных пользовательских интерфейсов вычислительных приложений, выполняющихся на суперкомпьютерах, актуальна для большого числа пользователей ВВС, разрабатывающих собственные программы для расчетов, и нетривиальна. Если бы программист создавал индивидуально некоторый интерфейс для сложного вычислительного приложения, то ему пришлось бы проделать довольно внушительную работу: написать скрипты запуска приложения под определённую вычислительную систему или несколько систем, если это требуется, также, если вычислительный эксперимент состоит из нескольких программ, то необходимо прописать скрипты взаимодействия программ и запуска их последовательно или параллельно, спроектировать и запрограммировать удобный интерфейс для работы с данным приложением. Данный процесс довольно трудоёмкий и требует знаний разработчика во многих областях разработки. В настоящей работе ставится задача избавить программиста от вышеперечисленных проблем с помощью сервиса, на который будет переложена большая часть проблем, возникающих в связи с задачей разработки пользовательского интерфейса приложений, работающих на ВВС.

Такой сервис должен уметь предоставлять унифицированный доступ к ВВС, а именно: позволять собирать приложение, отправлять его на выполнение на ВВС, отслеживать его состояние, передавать результат пользователю и т.д., кроме того позволять выполнять приложения в рамках сценариев, задающих порядок вызова приложений, передачи данных между приложениями, а также создавать необходимый для приложений пользовательский интерфейс для задания входных данных, просмотра и анализа результатов вычислений и мониторинга хода вычислений.

1.1. Обзор способов взаимодействия пользователей и/или интерфейсных систем с высокопроизводительными вычислительными системами

Существует множество различных типов высокопроизводительных вычислительных систем, на которых установлены различные и по-разному настроенные СУПЗ (системы управления прохождением задач) или же, наоборот, у некоторых систем СУПЗ отсутствует. Соответственно, для каждого типа системы запуск некоторой вычислительной задачи на такой системе будет отличаться. Рассмотрим несколько примеров запусков задач на высокопроизводительных вычислительных системах. Рассматриваемые ниже примеры взаимодействия с ВВС для удобства разобьем на группы:

- Системы без СУПЗ, например, любой Linux-сервер с необходимым системным программным обеспечением для запуска вычислительных задач;
- Системы с одной СУПЗ:
 - кластер НГУ;
- Системы с несколькими СУПЗ:
 - суперкомпьютер МВС-10П Межведомственного Суперкомпьютерного Центра Российской Академии Наук (МСЦ РАН);

Для начала рассмотрим примеры запуска задач на системах, не имеющих очереди задач.

Для запуска вычислительного эксперимента, написанного с применением библиотеки MPI [1], на одном из многоядерных Linux-серверов ИВМиМГ СО РАН необходимо, используя сетевой протокол ssh зайти на сервер в формате:

```
ssh <логин>@<адрес_сервера> -p <порт>
```

После выполнения данной команды требуется ввести пароль. Далее необходимо выбрать каталог, в котором хранится задача, требующая вычисления или создать новую задачу, после чего предварительно собранный исполняемый файл с

именем, например, *task.comp* можно запустить, используя команду *mpirun* вместе с необходимым набором параметров, в простейшем случае таковыми являются параметр указания количества потоков (*-np*) и максимальное время счёта (*-maxtime*), в более сложных случаях могут добавляться и другие параметры.

Пример выполнения команды *mpirun*:

```
mpirun -np 4 -maxtime 20 ./task.comp
```

Рассмотрим примеры запуска задачи на BBC, имеющей некоторую очередь задач.

Для запуска вычислительной задачи на *кластере НГУ* [2], необходимо, используя язык системы Altair PBS Pro (система управления прохождением задач в BBC [3, 4]) написать скрипт, например с именем *script.sh*, определяющий параметры задачи, см. Приложение А. После чего, используя команду *qsub* поставить написанный ранее скрипт в очередь:

```
qsub script.sh,
```

которая после проверки скрипта и помещения задачи в очередь выведет на экран строку вида

```
123456.hpc-suvir1.hpc,
```

где «123456» – число, являющееся уникальным идентификатором задачи.

После завершения работы задачи, в текущей директории появляются файлы с именами *script.sh.e123456* и *script.sh.o123456*, в которые перенаправляются стандартный поток ошибок и стандартный поток вывода соответственно, запускаемых в рамках задачи процессов параллельной программы.

Запустить вычислительную задачу удалённо на BBC можно также с помощью *MATLAB* [5, 6, 7]. К примеру, имеется программа, написанная на *MATLAB* и находящаяся на локальном компьютере и некоторый кластер, на котором есть возможность запуска данной программы (установлен *MATLAB*). Тогда, после ввода пользователем логина и пароля к BBC в локально установленном *MATLAB*, при запуске задачи локальный *MATLAB* обращается с по-

мощью протокола *ssh* на кластер и запускает программу на этой ВВС, посредством её СУПЗ, например PBS Pro.

Рассмотрим примеры запуска некоторой вычислительной задачи на удалённой высокопроизводительной системе, имеющей несколько различных СУПЗ.

Запуск задач на суперкомпьютере *MBC-10П МСЦ РАН* [8] можно осуществлять несколькими способами, в зависимости от типа поддерживаемых СУПЗ. Данный суперкомпьютер поддерживает 2 типа СУПЗ:

- СУППЗ (система управления прохождением параллельных заданий),
- планировщик SLURM.

Рассмотрим последовательно оба варианта запуска задачи на ВВС с СУПЗ.

Для запуска задачи с использованием СУППЗ необходимо предварительно загрузить модуль СУППЗ:

```
module load launcher/suppz
```

после чего выполнить рассмотренную выше команду *mpirun* для предварительно скомпилированной задачи с некоторым именем, к примеру, *task.comp* и необходимыми параметрами, например:

```
mpirun -np 4 ./task.comp
```

Данная команда автоматически формирует паспорт задания и направляет его в очередь СУППЗ. Просмотр очереди СУППЗ осуществляется командой

```
mqinfo [-s system],
```

где *system* – имя логической системы.

Запуск задачи с использованием планировщика SLURM можно выполнить в пакетном или интерактивном режимах.

В *пакетном* режиме пользователь создает скрипт для запуска задачи и использует утилиту *sbatch*. Задача отправляется в очередь и будет выполнена как только будут доступны запрошенные ресурсы. Вывод результата выполнения

задачи будет записан в файл *slurm- \langle номер задачи в очереди \rangle .out* в директории, откуда осуществлялся запуск задачи. В качестве параметра утилите *sbatch* передаётся исполняемый скрипт, который будет запущен на первом из выделенных вычислительных узлов. Внутри скрипта осуществляется запуск задачи с помощью одной из двух утилит: *srun* или *mpiexec.hydra*. Утилита *sbatch* позволяет задавать необходимые ей опции внутри скрипта запуска. Формат задания опций:

#SBATCH <опция>

Для запуска с помощью утилиты *srun* сперва должен быть загружен модуль *launcher/slurm*. Скрипт для пакетного запуска X экземпляров MPI-программы с именем, например, *task.comp* на каждом из Y вычислительных узлов приведён в приложении Б.

Для запуска с помощью утилиты *mpiexec.hydra* сперва должен быть загружен модуль *launcher/intel*. Скрипт для пакетного запуска X экземпляров MPI-программы с именем *task.comp* на каждом из Y вычислительных узлов приведён в приложении В.

В *интерактивном* режиме у пользователя есть два пути для запуска задачи:

- использовать для запуска задачи утилиту *srun*,
- самостоятельно выделить необходимое количество вычислительных узлов (ВУ) и запустить задачу с помощью утилиты *mpiexec.hydra* или *srun*.

Вывод результата выполнения задачи в обоих случаях будет произведен в консоль.

Пример запуска задачи с именем, например, *task.comp* с помощью утилиты *srun* (предварительно для корректной работы необходимо загрузить модуль *launcher/slurm*, если не загружен):

```
srun -N Y --ntasks-per-node=X -n X*Y ./task.comp,
```


где $-N$ – количество необходимых узлов, `--ntasks-per-node` – количество процессов запускаемых на каждом вычислительном узле.

Пример самостоятельного выделения ВУ и запуска задачи `task.comp` с помощью утилиты `mpiexec.hydra` (предварительно для корректной работы необходимо загрузить модуль *launcher/intel*, если не загружен):

Выделение Y вычислительных узлов с помощью команды *salloc*:

```
salloc -N Y,
```

где $-N$ – количество необходимых узлов. При наличии ресурсов будет выведено сообщение об успешном выделении вычислительных узлов, после которого необходимо использовать команду *mpiexec.hydra*:

```
mpiexec.hydra -perhost X -n X*Y ./hello_mpi_new,
```

где `-perhost <n>` – запуск n процессов на каждом узле, `-n` – общее количество процессов. Далее задача будет запущена, а результат выполнения будет выведен на экран.

Все рассмотренные выше способы взаимодействия с вычислительными ресурсами могут применяться к виртуальным вычислительным ресурсам, создаваемым с помощью облачных сервисов, таких как *Amazon Web Services (AWS)* или *Microsoft Azure*. Однако это означает лишь то, что к проблемам, описанным выше, добавляется ещё проблема организации управления виртуальными машинами.

Исходя из анализа приведенных выше систем организации вычислений на ВВС, можно сделать вывод о том, что проблема запуска прикладной задачи на ВВС решается, но решения этой проблемы зависят от типа ВВС и чаще всего требует изучения документации к выбранной ВВС. Однако некоторого универсального подхода, позволяющего запустить прикладную задачу на любой высокопроизводительной вычислительной системе без предварительного ознакомления с ней не существует. Отсюда вытекает необходимость в промежуточном программном обеспечении (ПО), которое позволит пользователям запускать прикладные задачи на различных вычислительных системах и при этом такое

ПО должно быть модульным для расширения числа высокопроизводительных вычислительных систем, на которых возможно запускать вычислительные задачи с помощью данного ПО. На основе анализа инструментария для управления задачами на различных ВВС, формируется требование к промежуточному ПО обеспечить унифицированный интерфейс для реализации следующих операций работы с ВВС:

- операция передачи данных на ВВС;
- операция запуска вычислительных задач;
- операция постановки в очередь некоторой системы управления прохождением задач (СУПЗ);
- операция отслеживания статуса выполнения запущенной задачи;
- операция извлечения результатов задачи;
- операция последующего анализа данных задачи.

В дальнейшем, операции из этого списка называются *базовыми операциями* работы с ВВС.

1.2. Формальное представление знаний

Выделяют различные парадигмы программирования, рассмотрим некоторые из них:

- императивное программирование – парадигма программирования, которая задаёт процесс исполнения некоторого математического алгоритма в виде инструкций вычислительной машины, при этом переменные алгоритма отображаются в ресурсные переменные – то есть в ячейки памяти и т.д.
- декларативное программирование – парадигма программирования, в которой задаётся спецификация решения задачи, то есть описывается проблема и ожидаемый результат, противоположно императивному программированию не указываются шаги достижения результата;

Для вычислительных задач разрабатываются декларативные языки, например, язык Норма [9, 10] в которых прикладному математику даётся возможность сформулировать свою задачу, например, записать некоторый численный метод решения задачи математической физики в привычных для него терминах, наподобие описания методов в различных учебниках и пособиях [11].

Если подобный метод из некоторого научного труда запрограммировать, используя императивный подход на каком-либо языке программирования, например на языке C++, то впоследствии, имея только запрограммированный алгоритм вычисления, невозможно определить автоматически, значения каких переменных алгоритма хранятся в тех или иных ячейках памяти вычислительной системы, в тех или иных позициях в выходных файлах вычислительного эксперимента [12].

Поэтому, если ставится цель автоматизировать работу математика-прикладника с высокопроизводительной вычислительной системой, необходимо предоставить для него возможность описывать алгоритмы в каких-либо более высокоуровневых терминах, нежели термины, используемые при императивном подходе программирования. В идеале, должна предоставляться возможность описания предметной области на математическом языке, синонимичная описаниям в научных работах.

Идея заключается в следующем: знания о предметной области вычислительной задачи необходимо выразить в формальном языке. В качестве примера можно использовать язык вычислительных моделей [11] или онтологий [13].

В таком случае, если имеется достаточно полное формальное описание предметной области, связанной с задачей, это позволяет на высоком уровне ставить спецификацию вычислительной задачи и получать автоматически сгенерированные алгоритмы, а в идеале — программы, которые выполняют эти алгоритмы и позволяют решать, таким образом, задачи, специфицированные на высоком уровне. Например, если есть формальное представление некоторого вычислительного метода, то его можно автоматически анализировать и на его

основе сгенерировать программу на каком-либо императивном языке программирования, тем самым избавляя математика-прикладника от написания такой.

Тем не менее, данный подход весьма трудно реализуется на практике, так как не всегда возможно достаточно полно описать предметную область задачи, а для нетривиальных предметных областей такое полное описание неосуществимо. Генерация программы по математическому алгоритму подразумевает необходимость решать задачу отображения операций и переменных некоторого математического алгоритма на ресурсы распределённой вычислительной системы, которая является NP-полной. Поэтому решения для таких задач можно находить либо в виде эвристик, либо в виде оптимальных решений для отдельных классов задач, что составляет отдельную научную проблему [11].

Эти вопросы до сих пор открыты, и эффективное решение пока не найдено. Тем не менее, исследования могут продвигаться на основе ограничения проблематики некоторыми частными предметными областями.

Помимо автоматической генерации программы по полному описанию предметной области некоторой задачи и спецификации задачи, есть возможность автоматической визуализации различных характеристик задачи.

Рассмотрим следующую ситуацию: пусть имеется некоторая серия массивов, тогда её можно визуализировать, если в формальном языке, в котором описана предметная область текущей задачи, будет предоставлена пользователю возможность сказать, что это серия массивов. Тогда к системе всегда можно будет добавить модуль, который проанализировав описание предметной области и найдя в ней описание понятия серии массивов, после окончания работы пользовательской программы и формирования этой серии массивов предоставит нам визуализацию этой серии. Сама по себе визуализация серии массивов - задача известная, например, для серии двумерных массивов может быть сгенерировано видео, где каждый кадры соответствует массивам серии, а цвет пикселей определяется значениями элементов массивов. Или, имея значения эле-

ментов массива, среди них можно отыскать максимальный и минимальный элементы, и установив некоторую цветовую градацию в соответствии с найденными элементами сгенерировать изображение, на котором будут отображаться значения в виде графика или некоторой поверхности.

Однако, если программа написана на некотором императивном языке программирования, например на языке C++, то не всегда известно что хранимые в памяти данные это именно необходимая нам серия массивов. Поэтому встаёт вопрос о том, чтобы позволить пользователю ввести некоторые понятия, то есть предложить такой декларативный язык, в котором пользователь вводит понятия, необходимые ему, и описывает их или использовать, например, язык OWL для описания онтологий [13]. Таким образом, если имеется формальное описание некоторого понятия, то можно автоматически сгенерировать представление этого понятия (например, проинициализированную соответствующую структуру данных или визуализацию), а также дополнить некоторыми операциями работы с данным понятием, при необходимости.

Так как сама по себе разработка такого языка для описания предметной области некоторой вычислительной задачи на языке близком к математическому языку довольно сложная задача, то для простоты вместо описания самого алгоритма, в рамках магистерской диссертации принято решение ограничиться описанием того, что имеет значение с внешней точки зрения относительно алгоритма – то, что подаётся ему на вход и то, что получается на выходе после прохождения данного алгоритма.

Помимо визуализации выходных параметров, рассмотренной выше, разрабатываемый язык должен включать в себя возможность описания области моделирования с последующей визуализацией. То есть пользователь на вход алгоритму может подать геометрию предметной области, в которой будет происходить моделирование или описания объектов, принадлежащих интересующей пользователя области, а система по предоставленной геометрии сама сформу-

лирует параметры, требуемые вычислительному алгоритму на вход или же визуализирует описанные объекты для заполнения их значений.

Кроме того, возникает необходимость накопления знаний о способах вычислений интересующих нас величин, хранящихся в системе, через уже имеющиеся данные, то есть об алгоритмах и нетривиальных вычислениях, требующих высокопроизводительных ресурсов. Использование онтологий позволит нам создавать интерфейсы для вычислительных приложений, работающих на некоторых высокопроизводительных вычислительных ресурсах, однако помимо интерфейса необходим и некоторый унифицирующий сервис для таких приложений. В качестве такого унифицирующего сервиса может выступать сервис HPC Community Cloud (HPC2C), позволяющий ставить вычислительные приложения на выполнение на удалённых высокопроизводительных вычислительных системах и удовлетворяющий базовым операциям работы с BBC, определёнными выше (см. раздел 1.1).

В данном сервисе с пользовательской стороны можно описать предметную область вычислительного приложения с помощью онтологий, а в дальнейшем на основе данного описания генерировать интерфейсы для конкретных приложений. При этом с вычислительной стороны реализовать или взять готовую вычислительную часть приложения и обеспечить её выполнение на тех или иных высокопроизводительных ресурсах. HPC Community Cloud содержит в себе необходимую информацию для предоставления пользовательского интерфейса в той или иной форме, то есть в своей основе он является API (сокращение от *application programming interface* – интерфейс программирования приложений) с помощью которого можно запрограммировать необходимый для конкретной цели интерфейс (мобильный интерфейс, веб-интерфейс и пр.) и кроме того выполнять указанные в интерфейсе действия на устройствах.

HPC Community Cloud (HPC2C) [14, 15] – программный инструментальный, объединяющий ресурсы различных высокопроизводительных вычислительных систем (BBC) в единый сервис и предоставляющий сторонним программным

системам и пользователям единую точку доступа для работы с этим сервисом. Также HPC2C является платформой для разработки и накопления с целью повторного использования программных решений прикладных задач, выполняющихся на ВВС.

HPC2C предоставляет среду для накопления знаний в виде фреймворков и моделей. Разрабатывать встроенные программные решения могут не только разработчики HPC2C, но и другие пользователи. Для достижения этой цели они могут использовать интегрированную среду разработки для редактирования файлов.

1.3. Исследование текущей версии HPC Community Cloud

Программное обеспечение сервиса HPC Community Cloud (HPC2C) на момент исследования состоит из сервера, предоставляющего интерфейс прикладного программирования RESTful (API) для внешних программных систем, и веб-приложения, предоставляющего графический интерфейс пользователям и упрощающий взаимодействие с API. Кроме того, API HPC2C является основой для разработки внешних программных систем, которые могут получить доступ к ресурсам компьютерных центров для крупномасштабных вычислений. Сервис скрывает особенности доступа к высокопроизводительным вычислительным системам за унифицированным доступом к имеющимся системам и позволяет добавлять новые ВВС указав необходимые характеристики

Рассмотрим подробнее принципы работы с Web-интерфейсом HPC2C и выделим некоторые особенности и недостатки сервиса. Для этого выполним исследование сервиса, согласно инструкции представленной ниже.

1. Зарегистрировать пользователя и/или войти. Требуется ввести логин и пароль в форму входа в систему, после чего нажать кнопку Sign in (рисунок 1).

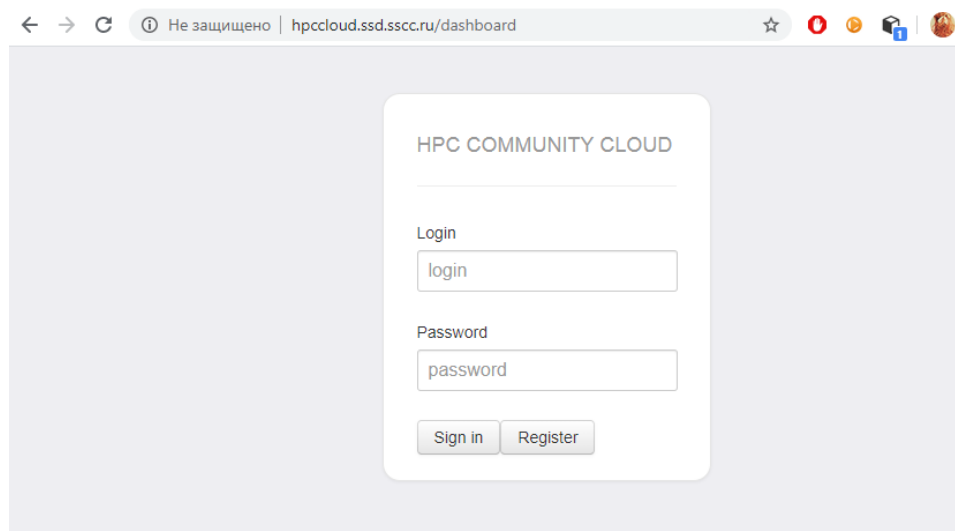


Рисунок 1 – Форма входа в HPC2C

Если пользователь отсутствует, то необходимо зарегистрировать его в системе, нажав на кнопку Register и заполнить предлагаемую форму (рисунок 2), после чего сохранить данные и войти в систему введя логин и пароль в форму, описанную выше.

The image shows a web browser window with the address bar displaying 'hpccloud.ssd.ssc.ru/register'. The page title is 'HPC COMMUNITY CLOUD'. The registration form contains the following fields: 'Login*' with the placeholder 'nstuproj', 'Password*' with a masked password '*****', 'Retype your password*' with a masked password '*****', 'First name' with the placeholder 'firstname', 'Last name' with the placeholder 'lastname', 'E-mail' with the placeholder 'your@mail.com', 'Phone' with the placeholder '+7 xxxxxxxxxx', and 'Access code' with a masked code '*****'. A 'Register' button is located at the bottom of the form. Below the button, there is a small asterisk and the word 'required'.

Рисунок 2 – Форма регистрации нового пользователя в HPC2C

2. Добавить собственный профиль удалённой вычислительной системы. После входа и/или регистрации пользователя в системе HPC2C он оказывается в личном кабинете пользователя (рисунок 3), где имеет возможность просматривать созданные ранее приложения в разделе Applications, фреймворки в разделе Frameworks, модели в разделе Models, запущенные ранее задачи с некоторым статусом в разделе Jobs и доступные вычислительные системы в разделе Clusters.

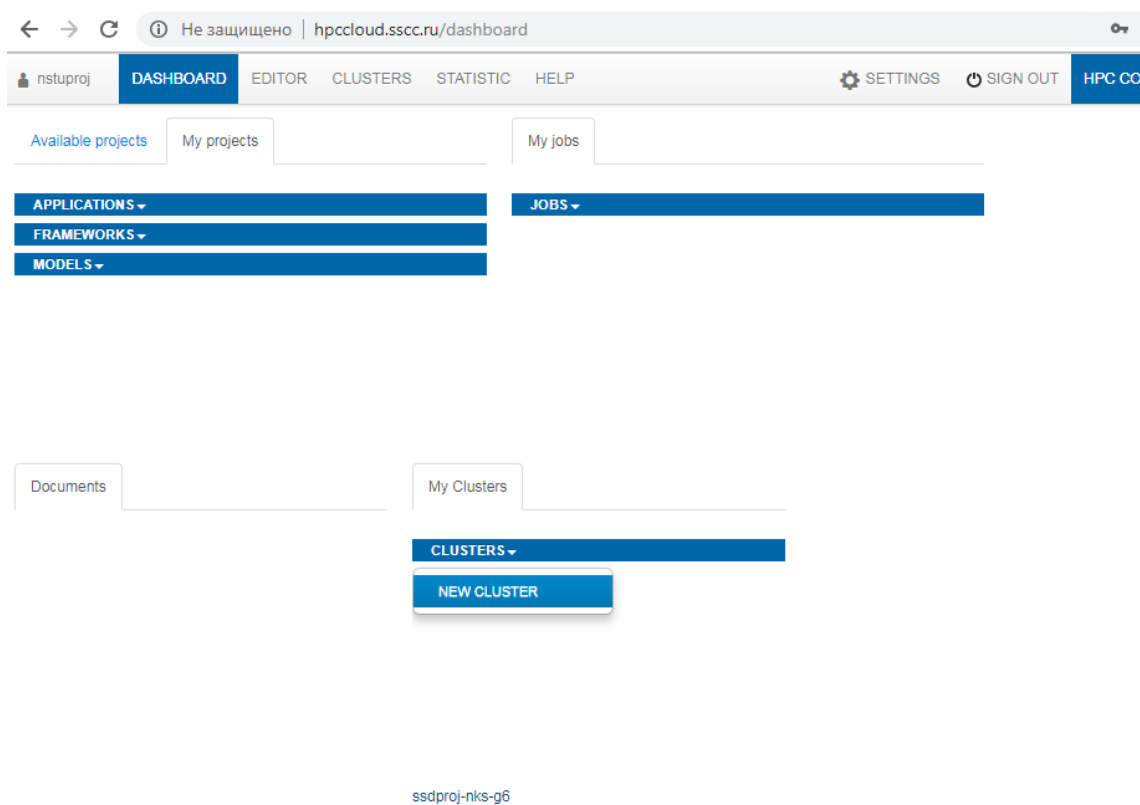


Рисунок 3 – Личный кабинет пользователя и кнопка для добавления новой вычислительной машины в систему

Для дальнейших экспериментов необходимо прописать доступ на некоторый удалённый сервер, например, можно воспользоваться учётной записью на сервере `hpccloud.ssd.sccc.ru`. Чтобы добавить профиль системы, необходимо в разделе Clusters выбрать пункт New cluster, и ввести данные учётной записи удалённого сервера (рисунок 4).

Add new cluster profile

Name:

Interface:

URL:

Login:

Password:

SSH private key:

Cancel Ok

Рисунок 4 – Форма для добавления профиля новой вычислительной машины в систему

3. Создать проект для разработки приложения. Проекты создаются на основе шаблонов, в настоящее время доступен шаблон MPI-программы и LuNA-программы. Для создания нового приложения необходимо в разделе Applications выбрать пункт New app (рисунок 5), после чего во всплывающем окне необходимо указать имя приложения, шаблон (template) и целевую вычислительную систему (cluster profile), в данном примере необходимо выбрать профиль созданного сервера: hpccloud.ssd.sccc.ru (рисунок 6).

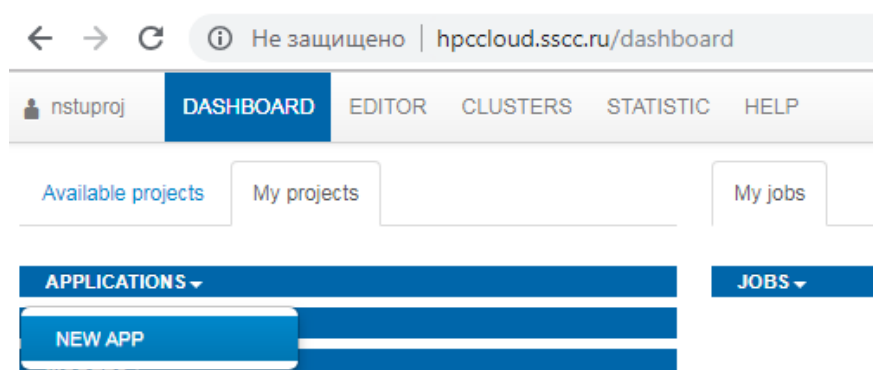


Рисунок 5 – Кнопка для создания нового приложения

Add new application

Name:
example01

Template:
mpicxx

Cluster profile:
hpccloud.ssd.sccc.ru

Make configuration:
release

Cancel Ok

Рисунок 6 – Форма для создания нового приложения

Далее в открывшемся окне необходимо прописать код приложения, который впоследствии можно неоднократно модифицировать, и сохранить его, нажав кнопку Save (рисунок 7).

4. Отправить разработанный код на сборку на целевой вычислительной системе. После написания кода, перед постановкой задачи на выполнение на ВВС, его необходимо собрать на выбранной ранее вычислительной системе, для этого требуется нажать кнопку Make (рисунок 7) и после непродолжительного ожидания можно увидеть ошибки компиляции или сообщение об успешном построении объектных файлов (рисунок 8).

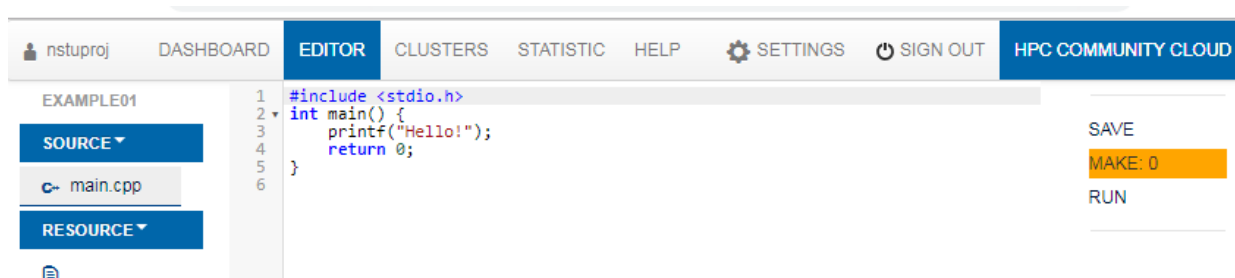


Рисунок 7 – Написание текста приложения и его сборка

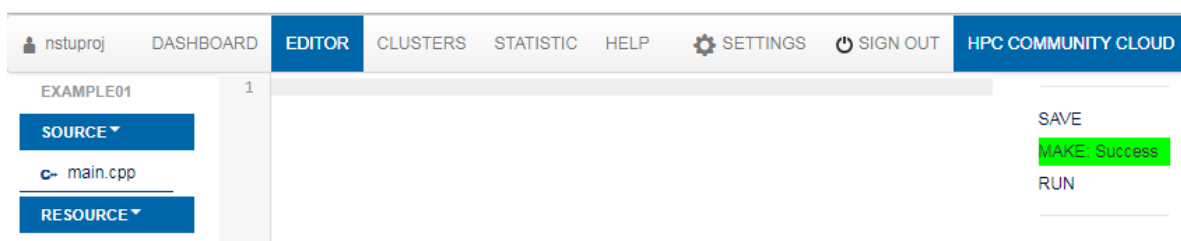


Рисунок 8 – Уведомление об успешной сборки приложения

5. Сформировать и отправить на удалённую вычислительную систему расчётную задачу на основе успешно собранного приложения. После успешной сборки приложения необходимо поставить задачу на исполнение на выбранной ранее ВВС, для этого необходимо нажать на кнопку Run (рисунок 8) и во всплывающем окне указать требуемые параметры запуска (рисунок 9), если ВВС является кластером, то требуется задать количество узлов и процессов на узел, а также оставить по умолчанию или выбрать очередь кластера, если же ВВС является обычным сервером, то задать только имя задачи (рисунок 9). Далее требуется нажать кнопку Ok и получив уведомление об успешной постановке задачи на исполнение вернуться в личный кабинет для отслеживания состояния выполняемой задачи (рисунок 9).

Рисунок 9 – Форма для постановки новой задачи на выполнение на ВВС и уведомление о начале вычислений

6. Отслеживать статус задачи на кластере через dashboard. В личном кабинете в разделе Jobs появится запущенная ранее задача с некоторым статусом. После завершения расчётов статус изменится на Finished[load results] (рисунок 10), после чего результаты расчётов начнут загружаться в систему и после завершения загрузки статус изменится на Finished (рисунок 10) и пользователь получит возможность просмотра результатов.

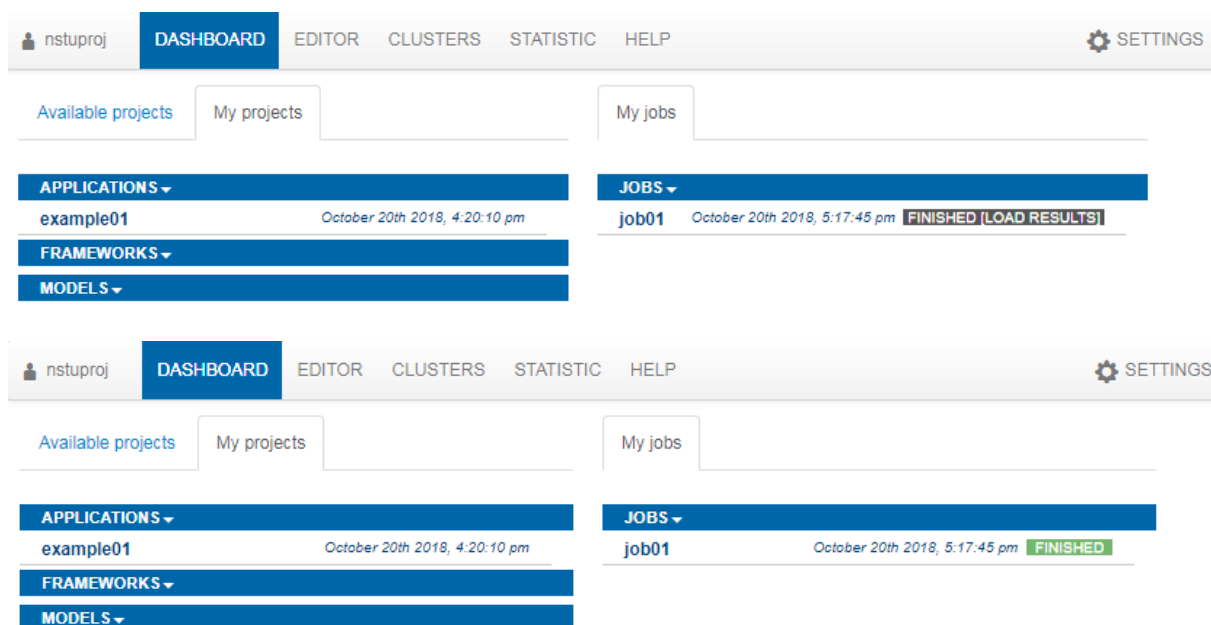


Рисунок 10 – Уведомления о выгрузке результатов работы приложения и завершении вычислений

7. В случае успеха или неудачи вычисления задачи просмотреть файлы с результатами. Для просмотра результатов требуется в разделе Jobs личного кабинета пользователя нажать на имя задачи, чтобы перейти к просмотру файлов вывода или ошибок, если таковы имеются (рисунок 11).

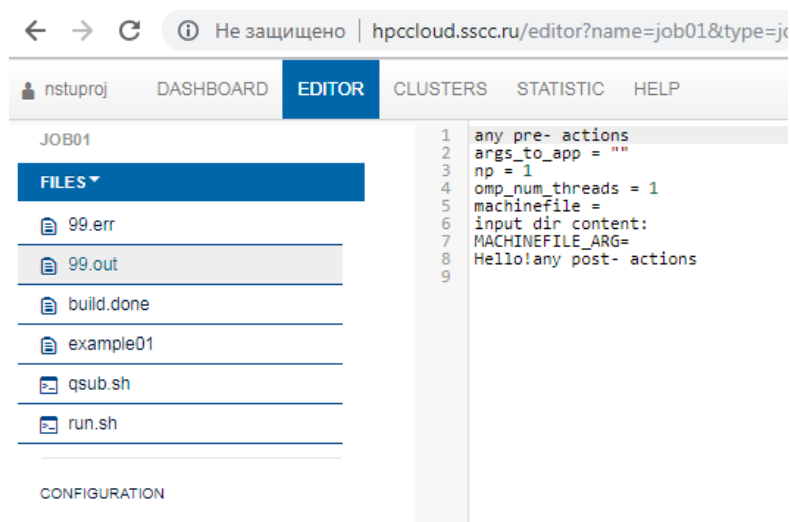


Рисунок 11 – Результат выполнения вычислений

В ходе исследования сервиса были выделены следующие достоинства сервиса HPC2C:

- сервис удовлетворяет базовым операциям работы с ВВС, определённым ранее;
- позволяет взаимодействовать с несколькими вычислительными системами разных типов;
- имеет пользовательскую авторизацию в системе, которая предоставляет безопасность данных внутри аккаунта;
- позволяет создавать и редактировать приложения внутри сервиса;
- сохраняет данные внутри аккаунта и имеет возможность накапливать их.

Также были выделены следующие недостатки в работе HPC2C:

- отсутствие регистрации пользователей через социальные сервисы (аккаунт Google, Vkontakte, Facebook и др.);
- отсутствие возможности разработки приложений совместно, а также предоставления доступа к приложению одному или нескольким пользователям;
- отсутствие возможности группировки приложений/задач по требуемым признакам и поиска на основе такой группировки;

- отсутствие автоматической генерации пользовательских интерфейсов к вычислительным приложениям, разрабатываемым под управлением HPC2C;
- отсутствие высокоуровневого управления данными вычислительных приложений;
- для сохранения изменений модификаций кода приложения необходимо нажимать кнопку SAVE неоднократно;
- для отслеживания статуса задачи на кластере необходимо вручную перезагружать личный кабинет сервиса.

1.4. Обзор систем для создания workflow и генерации интерфейсов

Интерфейсы генерируют давно, в том числе и с использованием онтологий. Так как онтология является по своей сути формальной моделью, которая показывает в каких отношениях находятся понятия предметной области, предлагается, в рамках магистерской диссертации, вместо онтологий [13] использовать вычислительные модели [11] как частный случай онтологий, поскольку в предлагаемом нами ниже подходе все отношения между понятиями предметной области имеют один вид, а именно: один объект можно вычислить через другой и при этом указать конкретную операцию (процедуру/программу/сервис) для реализации этого вычисления.

Идеи генерации интерфейсов с помощью спецификации входных и выходных данных рассматривались давно, причём генерация пользовательских интерфейсов и использование так называемых workflow в основном реализуются облачно и на коммерческой основе. Рассмотрим несколько таких систем более подробно для выявления сильных и слабых сторон систем.

Начнём обзор систем с системы машинного обучения от Google под названием *Tensorflow* [16]. Данная система использует для построения нейронной сети слои и модели, может работать на параллельных процессорах, вычисления

внутри системы выражаются через граф состояний, позволяет работать в облачной среде, например в AWS и других облачных системах. Однако некоторого общего интерфейса для взаимодействия не имеет, а также основная функциональность системы нацелена на машинное обучение. Данную систему можно позиционировать как API, с помощью которого возможно разработать необходимый для использования интерфейс.

Система *Orange workflow* [17] позволяет, используя визуальное программирование, строить алгоритмы в виде workflow, а также предоставляет пользователю графический интерфейс для анализа данных, большой набор виджетов для работы с данными, моделями, а также виджеты позволяющие визуализировать требуемые данные. Кроме того система имеет алгоритмы машинного обучения, различные модули для работы с данными и их оцениванием и обладает открытым исходным кодом. Однако данная система узко специфицирована на data mining и не предоставляет возможности отправки данных на некоторые внешние ресурсы.

Аналитическая платформа *Loginom* [18], предназначенная для реализации аналитических процессов, как и предыдущая система позволяет скрыть сложность разработки алгоритмов и их логики за визуальными компонентами, предоставляет возможность работы с большими данными, имеет большой набор библиотек для работы, а также большой выбор средств визуализации данных и предоставляет возможность параллельного выполнения сценариев и расчётов. Недостатком данной системы является то, что система распространяется на коммерческой основе и не имеет возможности использования внешних суперкомпьютеров.

Планировщик *ProActive Workflows & Scheduling* [19] от компании Activeeon предоставляет Web-интерфейс для проектирования приложений, а также большой набор библиотек и наличие планировщика нагрузок, позволяющего увеличить эффективность работы алгоритма. Имеет возможность, помимо последовательного выполнения ветвей алгоритма выполнять его параллельно,

кроме того предоставляет возможность запуска в облаке, позволяет объединять требуемые ресурсы в пул ресурсов, однако к частным вычислительным системам доступа у системы нет, а также программное обеспечение предоставляется на коммерческой основе.

Среда обработки данных и визуализации *Luna* [20] предоставляет постоянно растущую библиотеку хорошо настроенных компонентов, позволяет реализовывать логику алгоритмов в виде вложенных графов и визуализировать данные, а также предоставляет возможность создавать пользовательские типы, является расширяемой средой с возможностью накопления библиотек. *Luna* – полностью функциональный WYSIWYG-язык для обработки данных, что расшифровывается как What You See Is What You Get – что видишь, то и получишь. Наличие двух эквивалентных синтаксических представлений (визуального и текстового) позволяет проследивать связи между объектами на разных уровнях абстракции и легко переключаться между ними. Отсутствует возможность запуска алгоритмов на суперкомпьютерах.

Подробнее рассмотрев данные системы, были выявлены критерии оценки систем относительно удобства использования системы и исходя из оценки данных критериев (таблица 1) можно сделать вывод, что ни одна из рассмотренных систем полностью не удовлетворяет поставленным критериям. Среди рассмотренных систем можно выделить две наиболее подходящие для решения поставленной нами ранее проблемы: платформу *Loginom* и среду обработки данных и визуализации *Luna*, однако и эти системы имеют свои недостатки, а именно не предоставляют возможности взаимодействия пользователя с различными высокопроизводительными вычислительными системами.

Таблица 1 – Исследование преимуществ и недостатков различных систем

Название системы	Наличие workflow	Визуализатор данных	Работа с внешними кластерами	Позволяет создавать собственные типы	Визуальное программирование	Широкий спектр возможностей	Открытый код	Бесплатный доступ
Tensorflow	—	+	—	—	—	+/-	+	+
Orange workflow	+	+	—	—	+	—	+	+
Loginom	+	+	—	—	+	+	—	—
ProActive Workflows & Scheduling	+	—	+	—	+	—	—	—
Luna	+	+	—	+	+	+	+	+

1.5. Формулирование требований к системе

1.5.1. Описание основных требований к системе

Сервис HPC2C в базовом варианте решает проблему унифицированного доступа к высокопроизводительным вычислительным системам и удовлетворяет базовым операциям работы с ВВС, введённым в первом разделе обзора. Однако понимания того какая именно задача запускается и что за данные подаются ей на вход и получаются на выходе нет, отслеживание этой информации возложено на конечного пользователя сервисом, что не всегда удобно, так как зачастую пользователи стремятся к упрощению своих обязанностей, задавая лишь необходимые значения и не заботясь о дальнейших пунктах, необходимых для корректного выполнения вычислительного эксперимента вплоть до получения результатов.

Следовательно, необходимо поднять уровень сервиса, относительно базового, научив систему отслеживать смысл входных и выходных данных и на более высоком уровне управлять самими задачами, в том числе связывать их в сложные системы, так называемые “workflow”. Одним из способов реализации данного уровня является создание некоторой системы для реализации workflow в виде вычислительных моделей, позволяющей описывать этапы разработанного вычислительного эксперимента более понятным для пользователя языком.

Смысл же введённых и полученных данных предполагается отслеживать на уровне языка спецификации, он позволяет нам понять последовательность действий, а также то, какие объекты данных получаются.

Возможность запуска задач на BBC остаётся, но с такой системной надстройкой появляется возможность более удобно задавать входные и выходные данные вычислительных приложений за счёт генерации интерфейсов для таких данных. Кроме того, использование workflow позволит нам оперировать более сложными задачами: связывать несколько экспериментов или их частей, если эксперимент состоит из нескольких различных программ, в одну общую workflow, более точно прорабатывать все ветви сложного приложения, создавать сложные схемы, подразумевающие неоднократное обращение к сервису HPC2C.

В итоге генерация пользовательских интерфейсов приложения – следующий этап после описания вычислительного эксперимента в виде workflow, так как сама workflow не выглядит как конечное приложение, которым удобно пользоваться, для пользователя. Некоторым пользователям нет необходимости знать как взаимодействуют и работают компоненты workflow, им нужно лишь указать входы приложения и получить выходы. Поэтому генерация интерфейсов приложения позволяет оставить на виду лишь осмысленные для пользователя задание входных величин и получение результата, так как все промежуточные этапы скрыты внутри сложной многоуровневой системы и это упрощает взаимодействие пользователя с вычислительным приложением.

1.5.2. Варианты взаимодействия пользователя с системой

Для конечного пользователя интерфейс для взаимодействия с некоторым сервисом может быть использован на разных платформах и в различных ситуациях, в которых находится пользователь, поэтому полное обслуживание пользователя состоит в том, что на всех направлениях, где для использования серви-

са по назначению хорошо было бы иметь интерфейс, необходима возможность разработки такого интерфейса.

К примеру, для работы с сервисом на настольном компьютере имеет смысл реализация десктопного приложения, так как оно интегрировано в операционную систему нативно. Основным преимуществом такого подхода является удобный доступ десктопного приложения к файлам файловой системы пользователя. Кроме того, десктопные приложения, такие как Dropbox или Google Drive, реализуются как приложения для операционной системы, что позволяет встраиваться в операционную систему и делать синхронизацию файлов с облаком и локальными файлами, которая недоступна браузерным версиям приложений. Кроме того ещё одним преимуществом десктопной реализации является возможность встраивания в контекстное меню, то есть при нажатии внутри приложения на правую кнопку мыши может появиться меню, связанное с данным приложением.

Если рассматривать реализацию некоторого Web интерфейса для работы с сервисом, то можно выделить следующие преимущества: отсутствие необходимости установки программного обеспечения на компьютер пользователя, доступ к системе практически из любого места, где есть доступ в Internet и браузер, а также нет зависимости от операционной системы пользователя, так как вся работа происходит непосредственно в браузере, причём от вида браузера реализация Web интерфейса также не зависит. Кроме того, пользователь при желании может использовать данный интерфейс и с других платформ, однако это может быть неудобно.

Преимущества мобильной реализации интерфейса для работы с некоторым сервисом заключаются в том, что мобильная реализация может использовать средства операционной системы, как и десктопная реализация, также синхронизировать облачные файлы с файлами пользователя из файловой системы. Кроме того, нет необходимости держать браузер открытым и путаться во вкладках мобильного браузера, так как в основном идёт работа с окнами интерфейса, при-

чём они более легковесны, чем браузерные или же десктопные и, в отличие от последних, не вмещают в себя много лишней информации, на каждом окне присутствует только то, что необходимо пользователю в данный момент. Дополнительным преимуществом такой реализации является возможность получения пользователем уведомлений по окончании некоторых процессов, протекающих внутри сервиса, что позволяет контролировать их и своевременно вносить изменения, если таковые требуются.

Таким образом, если есть некоторый сервис, то имеется необходимость, чтобы пользователи, находясь в некоторой среде, например, в поездке, дома за рабочим компьютером или за мобильным устройством, имели максимально удобный доступ к этому сервису, то есть интерфейс должен находиться в том канале, что доступен пользователю на данный момент. То есть должно происходить отслеживание того, где пользователь находится в данный момент и на основе спецификации адаптировать и сгенерировать интерфейс для взаимодействия с пользователем под канал, которым пользователь может воспользоваться.

Однако на практике сложно представить как такие интерфейсы будут появляться в любом канале, где находится пользователь и находить способ с ним взаимодействовать, следовательно, разработчики в соответствующем канале должны предусмотреть решение данной проблемы. В таком случае если пользователь обращается через мобильное устройство, то должно быть заранее подготовлено мобильное приложение, если обращение к сервису происходит через браузер – имеется Web интерфейс, а если же обращение идёт посредством настольного компьютера, то необходим десктопный интерфейс. Таким образом, будет обеспечено присутствие во всех каналах с помощью заранее подготовленных разработчиками интерфейсов.

1.5.3. Постановка задачи

В заключение обзора и исходя из рассмотренных выше требований к разрабатываемой системе можно сделать вывод, что поставленные ранее проблемы унификации доступа к ВВС, создания некоторого декларативного языка для описания предметных областей задач и проблема генерации интерфейсов входных и выходных данных приложения на основе их формальной спецификации решаются, но для очень узких областей и конкретных систем. Однако же, если начать рассматривать данные проблемы шире и в совокупности, то универсального решения такой общей проблемы пока не существует. В рамках магистерской диссертации ставится задача практически исследовать возможность решения этой проблемы на основе создания программной системы, в соответствии со следующими требованиями:

- разрабатываемая система должна обеспечивать унифицированный доступ к ВВС и удовлетворять базовым операциям работы с ВВС, введённым в первом разделе обзора;
- разрабатываемая система должна предоставлять пользователю подсистему для реализации workflow в виде вычислительных моделей и исполнения сложных сценариев;
- система должна иметь декларативный язык, позволяющий по формальной спецификации входных и выходных параметров генерировать удобные пользовательские интерфейсы для задания параметров задачи и для анализа результатов расчётов, посредством визуализации;
- система должна предоставлять пользователю оптимальный интерфейс для взаимодействия с системой, который соответствует той среде, в которой пользователь находится в конкретный момент времени;

- разрабатываемая система должна сохранять всю получаемую информацию от пользователей и вычислительных задач, а также иметь возможность отобразить её пользователю при соответствующем запросе.

2. ПРОЕКТ

Основная цель проекта – разработка инструментария для генерации пользовательских интерфейсов, следовательно, одной из ключевых проблем работы является понимание того, что за интерфейсы необходимо генерировать, как они должны быть устроены и какие компоненты должны в них находиться. Кроме того, остаётся актуальной проблема анализа типичных сред работы пользователей и реализации интерфейсов в соответствии с этими каналами.

2.1. Описание инструментария для разработки интерфейсов

Для реализации пользовательских интерфейсов взаимодействия с суперкомпьютерами необходима поддержка типичных операций взаимодействия с BBC. Необходимо, чтобы любой пользователь разрабатываемого сервиса мог обратиться через интерфейс к сервису, мог написать с его помощью вычислительное приложение, умел задавать входные данные для такого приложения, ставить задачу с заданными входными данными на выполнение на некоторой вычислительной системе, отслеживать статус её выполнения и получать, а также просматривать результаты выполнения задачи. Для такой последовательности действий можно воспользоваться сервисом HPC Community Cloud (HPC2C), который предоставляет API, с помощью которого можно запрограммировать необходимый для конкретной цели пользовательский интерфейс, либо воспользоваться готовым Web-интерфейсом HPC2C для управления выполнением приложений на BBC. Сервис предоставляет унифицированный доступ разрабатываемым внутри него приложениям к вычислительным системам.

Однако, если пользователю необходима более сложная последовательность действий, например, неоднократный запуск одной или нескольких задач, предварительная подготовка файлов для задачи, визуализация результатов и прочее, то предоставленной функциональности Web-интерфейса HPC2C будет недостаточно, так как число операций, которые требуется делать пользователю, и их сложность возросли, что сказывается на усложнении взаимодействия с

Web-интерфейсом. Это приводит к тому, что необходима разработка вычислительных сценариев и их поддержки внутри HPC2C, так как работа пользователя теперь состоит не только в простейшей постановке задачи на вычисление, ему необходимо выполнять некоторые относительно сложные сценарии вычислений. Для этого можно воспользоваться системой Ака: интерфейсной системой для работы с вычислительными моделями, под управлением HPC2C.

При постановке задачи на выполнение необходимо, для более удобного задания входных и выходных данных, по некоторой спецификации этих данных сгенерировать интерфейс, позволяющий пользователю задать значения входных переменных и после выполнения вычислений проанализировать результаты. С помощью API можно разработать такой специальный интерфейс для постановки любой конкретной задачи на выполнение [21]. При использовании вычислительных сценариев для выполнения приложений на BBC, пользователю также необходим некоторый интерфейс для задания входных и визуализации выходных данных. Так как сценарий предполагает наличие величин, являющихся входами и выходами для сценария, то на основе их спецификации можно сгенерировать интерфейс.

Входными и выходными величинами некоторого сценария могут быть объекты определённых типов, причём такие типы могут быть разными и довольно сложными, следовательно, возникает необходимость уметь конструировать такие объекты разных типов и на основе спецификации типов генерировать интерфейсы для визуализации результатов работы сценария и задания входных величин.

Для решения проблемы отсутствия интерфейсов пользователя в различных средах, под которыми подразумеваются основные платформы для работы пользователя: десктопная среда, браузерная среда и мобильная среда, предлагается разработать такие интерфейсы для основных сред, в которых может работать пользователь: браузер, мобильное приложение и стационарный компьютер. В рамках магистерской диссертации рассматривались браузерная среда и мо-

бильная среда и, соответственно, интерфейсы к данным средам. Реализация десктопного интерфейса является следующим этапом разработки и не рассматривается в данной работе.

Исходя из вышеперечисленных рассуждений, все предложенные выше решения можно объединить в общий инструментарий для генерации пользовательских интерфейсов и условно разделить инструменты на три группы:

- API, с помощью которого можно реализовать все недостающие комплексу компоненты;
- базовые или готовые компоненты, которые пользователь может использовать в своей работе: Web-интерфейс HPC2C, интерфейсная система Aka и мобильное приложение, предоставляющее доступ конечному пользователю к некоторым функциям API;
- инструменты для генерации специфических элементов графического интерфейса для конкретных приложений, а именно конструктор спецификации типов, генератор форм для задания значений объектов специфицированных типов, и визуализаторы данных, под которыми подразумевается набор готовых плагинов, разработанных пользователями и/или заложенных в систему, которые соответствуют тому или иному типу из базы типов и позволяют визуализировать данные с таким типом предложенным способом. В тех ситуациях, где данных инструментов недостаточно необходимо использовать API для разработки требуемого продукта.

2.2. Базовая функциональность системы

2.2.1 Описание интерфейсной системы Aka

В рамках сервиса HPC Community Cloud (HPC2C) в ИВМиМГ СО РАН реализуется система Aka: интерфейсная система для работы с вычислительными моделями, которая применяется для построения сложных вычислительных сценариев и постановки задач на вычислительных моделях.

Интерфейсная система Ака позволяет строить сценарии с помощью двух типов объектов, используемых при построении: переменная и операция. Объект “переменная” обозначается зелёным цветом (рисунок 12а)) и представляет некоторые данные, используемые в сценарии в качестве входных, выходных или промежуточных. Объект “операция” обозначается чёрным цветом (рисунок 12б)) и представляет некоторую процедуру, которая может быть использована для вычисления значений её выходных переменных по значениям входных. Связь между объектами выражается стрелкой, которая явно определяет зависимости переменных от операций (рисунок 12в)).

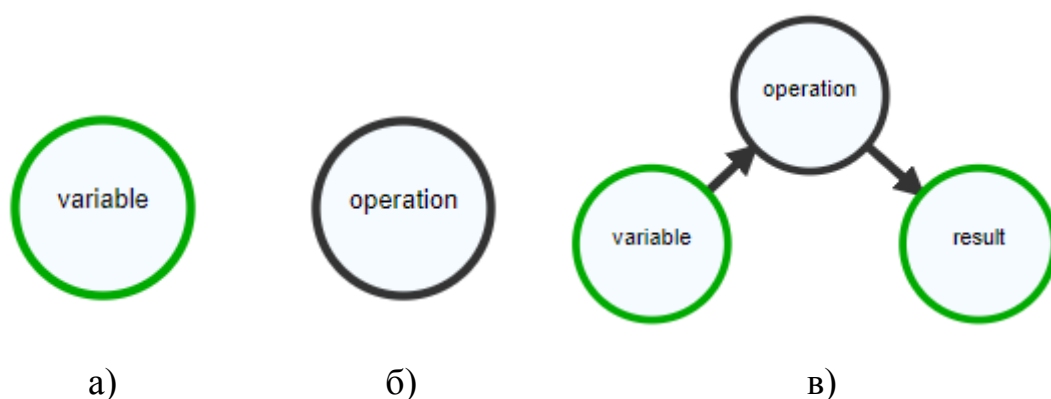


Рисунок 12 – Объекты интерфейсной системы Ака: переменная (а), операция (б) и связь между ними (в)

Построенный сценарий можно перемещать по экрану с помощью зажатия левой клавиши мыши, а также с помощью средней клавиши приближать или отдалять объекты сценария. Дополнительной функцией, упрощающей работу с объёмными моделями, является функция зуммирования отдельной переменной или операции посредством наведения на неё мышкой и нажатия клавиши Z, отвечающей за увеличение размеров элемента, что позволяет видеть название элемента при сильном уменьшении модели.

При необходимости пользователь может добавить дополнительные параметры переменной, например, задать тип переменной, определить её значение и так далее, с помощью модального окна (рисунок 13а)), вызываемого двойным щелчком по переменной. Модальное окно операции (рисунок 13б)) позволяет

вписать вычислительный алгоритм в данный объект и специфицировать локальные параметры, если операция является некоторой внешней моделью. Вызов модального окна операции аналогичен вызову модального окна переменной.

The figure shows two modal windows from the Aka system. Window (a) is titled 'variable' and contains the following fields: 'title' (value: variable), 'x' (value: 2195.76416015625), 'y' (value: 593.6800537109375), and 'value' (value: 200). There is a '+ =+' icon and an 'OK' button. Window (b) is titled 'operation' and contains the following fields: 'title' (value: operation), 'x' (value: 2152), 'y' (value: 621), 'type' (value: javascript), and a 'javascript' code block with the text 'result = variable / 2;'. There is a '+ =+' icon, a 'Variable mapping' section with 'variable' and 'result' fields, and an 'OK' button.

а)

б)

Рисунок 13 – Виды модальных окон интерфейсной системы Ака: модальное окно для переменной (а) и операции (б)


Для постановки вычислительной задачи, описанной в виде сценария, на выполнение с помощью системы Ака необходимо внутри сценария указать входные и выходные переменные из числа доступных переменных. Выбранные входные переменные обозначаются длинным жирным пунктиром, а выходные переменные – коротким и тонким (рисунок 14). Для выполнения вычислительного сценария достаточно указать значения входных данных сценария внутри модальных окон переменных и запустить выполнение модели с помощью кнопки , после чего пользователь сможет наблюдать выполнения последовательности действий внутри сценария.



Рисунок 14 – Подготовка вычислительной задачи к выполнению

По завершении расчётов внешнее обозначение входных и выходных переменных изменится (рисунок 15) и пользователю будет предоставлена возможность просмотра результатов с помощью модальных окон, где будут отображены значения выходных переменных, а также, если для этого имеется соответствующий модуль, появится возможность визуализации данных.

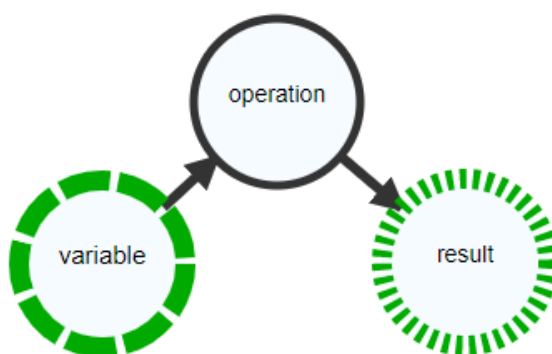


Рисунок 15 – Вид переменных после окончания вычисления

В рамках проекта в интерфейсную систему Ака предлагается добавить возможность генерации интерфейсов для задания значений входных переменных по их описанию. Для этого предлагается, указав внутри вычислительного сценария входные и выходные переменные, специфицировать типы этих переменных и сохранить полученное множество объектов как проблему (сформулированную задачу), указав её название и сохранив данные с помощью кнопки *Save as problem* (рисунок 14), которая сформирует ссылку на данную проблему и поместит на вкладку со всеми доступными задачами. После подготовки сценария к запуску в виде некоторой проблемы, на вкладке “Задачи” появится

сформированная ранее ссылка, которую следует выбрать, после чего пользователю будет предоставлен автоматически сгенерированный интерфейс для спецификации входных величин и последующей отправки задачи на выполнение. Заполнив предоставленные сгенерированные поля и имя будущего запуска необходимо отправить задачу на выполнение с помощью кнопки Solve Problem, после чего пользователь автоматически перейдёт на страницу с вычислительным сценарием, где будет наглядно показан процесс вычисления переменных, в процессе которого входные и промежуточные переменные, если они имеются, получают свои значения.

В процессе разработки и улучшения построенных пользователем сценариев они разрастаются и становятся достаточно массивными, число связей и возможных комбинаций действий увеличивается и результат уже сложно назвать сценарием, так как он содержит в себе несколько возможных сценариев вычислений величин. Таким образом, для сложных схем взаимосвязи переменных с операциями более применим термин вычислительная модель [22], которая позволяет автоматически генерировать требуемые интерфейсы пользователям. Человеку не приходится заново рисовать сценарии под каждый вариант вычисления переменных, ему достаточно в вычислительной модели, которая описывает предметную область, явно указать что является входными величинами в разных ситуациях и выходными, а после сохранить их как несколько проблем. Далее пользователю достаточно выбрать одну из сохранённых ранее проблем и для каждой из них будет автоматически сгенерирован интерфейс. Такой способ генерации интерфейсов можно использовать в качестве элемента инструментария для генерации интерфейсов.

Следует отметить, что вычислительная модель и визуальный конструктор, предоставляющий помимо варианта конструирования модели вариант использования его в качестве средства просмотра, их также можно использовать как часть интерфейса. То есть пользователь с помощью данного конструктора может проследить по визуализированной модели как были связаны величины и

последовательность работы некоторого алгоритма после выполнения вычислений.

2.2.2 Проект мобильного приложения HPC Community Cloud

Для удобства отслеживания выполнения поставленных задач, а также большей мобильности доступа к HPC2C был спроектирован мобильный интерфейс системы, предоставляющий пользователю приложения доступ к HPC2C и его основным функциям, необходимому пользователю для работы с сервисом. Смоделируем действия пользователя внутри мобильного приложения, предоставляющего доступ к сервису HPC2C. Сперва пользователь попадает на страницу входа в систему (рисунок 16а)), где ему предлагается ввести логин и пароль в предложенные формы для входа в систему, после чего подтвердить введенные данные, нажав кнопку Sign in.

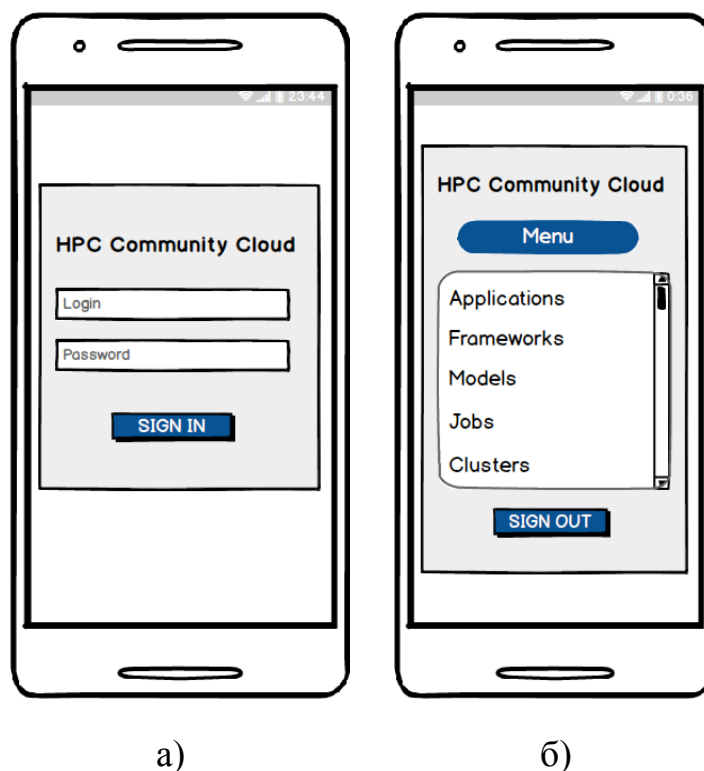


Рисунок 16 – Окна авторизации в системе (а) и меню приложения (б)

После авторизации пользователь попадает в личный кабинет (рисунок 16б)), где ему предоставляется возможность, выбрав из списка меню соответствующий пункт, просмотреть список доступных приложений (рисунок 17а)),

состояние доступных для работы кластеров (рисунок 17б)), список доступных для работы моделей (рисунок 17в)), список доступных фреймворков (рисунок 17г)), возможность просмотра состояния запущенных ранее задач (рисунок 18а)) и просмотра файловой системы пользователя (рисунок 18в)). Помимо интересующей пользователя информацией каждое окно оснащено кнопкой Menu, позволяющей выйти в личный кабинет пользователя, кроме того внизу большинства окон располагается кнопка выхода из приложения Sign out, позволяющая пользователю завершить работу с приложением и оказаться в окне авторизации.

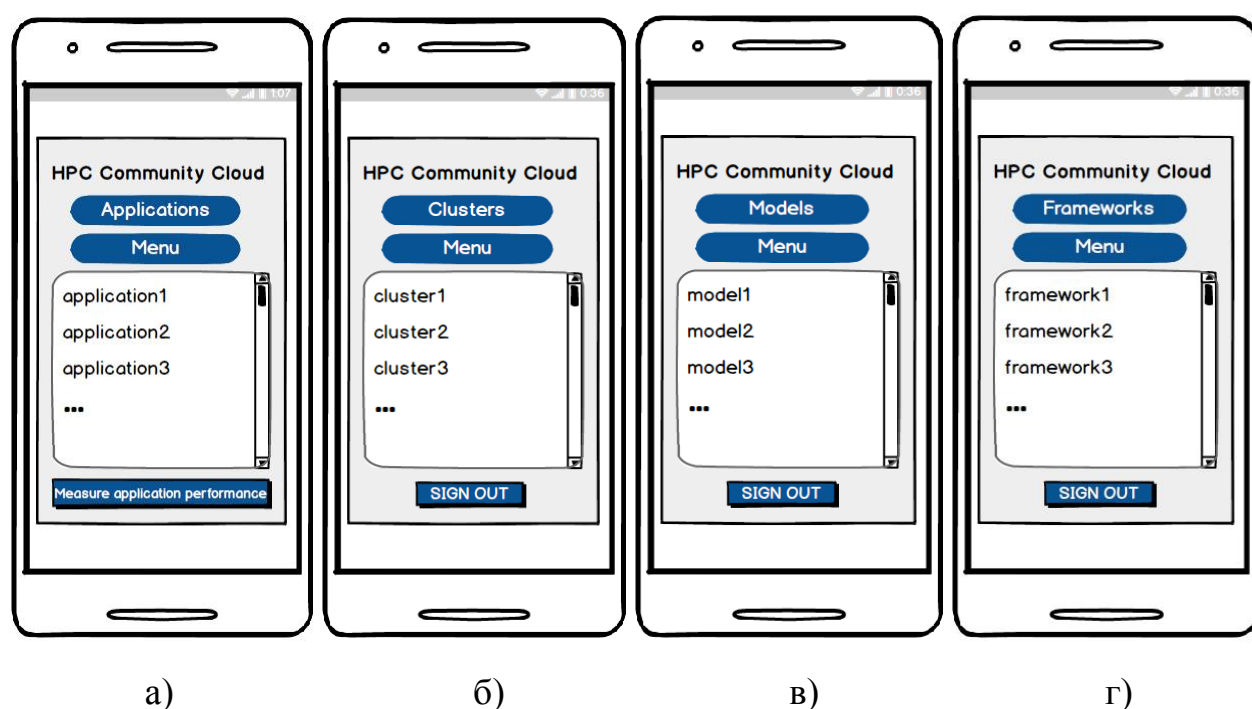


Рисунок 17 – Вид окон для элементов списка меню

При выборе из списка меню (рисунок 16б)) позиции просмотра состояния запущенных ранее задач пользователю предоставляется список этих задач с текущим состоянием (рисунок 18а)), кроме того пользователю также доступны функции обновления списка задач с помощью кнопки внизу окна и постановка новой задачи, с помощью кнопки New job (рисунок 18а)).

При нажатии на данную кнопку пользователь переходит в окно заполнения параметров, где ему необходимо ввести основные параметры, необходимые для

запуска, такие как имя задачи, количество узлов и процессов на один узел, а также определить приложение, которое в дальнейшем будет запущено и вычислительную систему, на которой этот запуск будет осуществлён (рисунок 18б)). После заполнения необходимых параметров пользователю необходимо нажать на кнопку Start и после всплывающего сообщения пользователь попадает на страницу со списком задач, где новая задача будет отображена со статусом “run” или “prepared”. Для отображения результатов задачи необходимо, чтобы её статус был “finished”, для смены статуса необходимо обновить список по кнопке, расположенной слева от кнопки New job. После того как статус задачи сменится на “finished”, при клике на имени задачи пользователь перейдёт на страницу с результатом выполнения задачи.

При выборе из списка меню позиции просмотра файловой системы (рисунок 18в)) пользователю предоставляется список доступных ему директорий и файлов с простейшей навигацией по папкам вперёд и возвращением на предыдущий каталог по стрелке, расположенной внизу окна.

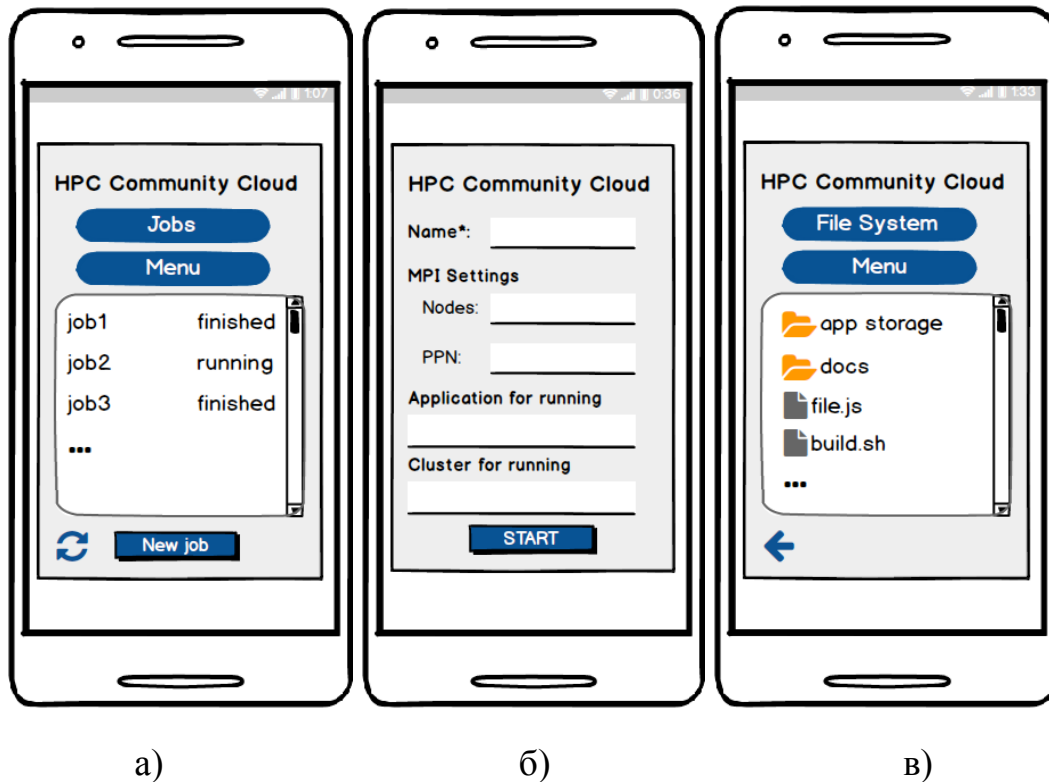


Рисунок 18 – Окна списка поставленных задач (а), постановки задачи на выполнение (б) и просмотра файловой системы (в)

Таким образом, спроектированный мобильный интерфейс для управления сервисом HPC2C охватывает большую часть функциональности базовой версии сервиса, предоставляя пользователю возможность как запуска новых задач на существующих вычислительных системах, так и просмотр результатов вычислений и имеющихся в аккаунте данных. Помимо предоставленных на данный момент возможностей интерфейс должен быть расширяемым, что позволит увеличить его функциональность посредством добавления новых модулей, отвечающих за те или иные функции работы с сервисом.

2.2.3 Описание профилировщика под управлением HPC Community Cloud

Профилирование — сбор характеристик работы программы, таких как время выполнения отдельных фрагментов (обычно подпрограмм), число верно предсказанных условных переходов, число кэш-промахов и так далее. Инструмент, используемый для анализа работы, называют профилировщиком или профайлером. Обычно выполняется совместно с оптимизацией программы. Часто используется, чтобы определить, как долго выполняются определённые части программы, как часто они выполняются, или для генерации графа вызовов.

Наличие профилировщика в сервисе позволяет пользователям сервиса оценивать производительность своих приложений по полученным от профилировщика данным, следовательно, пользователям нет необходимости реализовывать собственный профайлер. В рамках работы под оценкой производительности понимается постановка на выполнение определённого приложения с изменяющимся количеством ядер для запуска, последующим сбором результатов с этих запусков и выводом собранных данных в удобном для запрашивающего виде. Кроме того, в рамках работы профилирование рассматривалось как дополнительно встроенный модуль в мобильный интерфейс доступа к функциональности HPC2C.

Для того чтобы воспользоваться данным модулем, необходимо в меню приложения выбрать из списка (рисунок 16б)) позицию просмотра доступных

пользовательских приложений, после чего пользователь имеет возможность оценить производительность своего приложения, нажав на кнопку Measure application performance (рисунок 19а)), расположенную под списком приложений. После нажатия на кнопку пользователь попадает в окно профилировщика, где необходимо ввести данные, такие как общее имя для последующих запусков, минимальное и максимальное количество ядер для запуска, а также выбрать приложение для оценки и нажать кнопку Start (рисунок 19б)). После начала работы профилировщика необходимо ожидать пока он не закончит выполнение и после всплывающей подсказки нажать на кнопку See result, которая открывает окно, где будут представлены результаты оценки производительности приложения в виде таблицы, содержащей в себе информацию о количестве ядер для каждого запуска вычислительного приложения, времени вычисления задач, эффективности и ускорению приложения относительно первого запуска (рисунок 19в)).

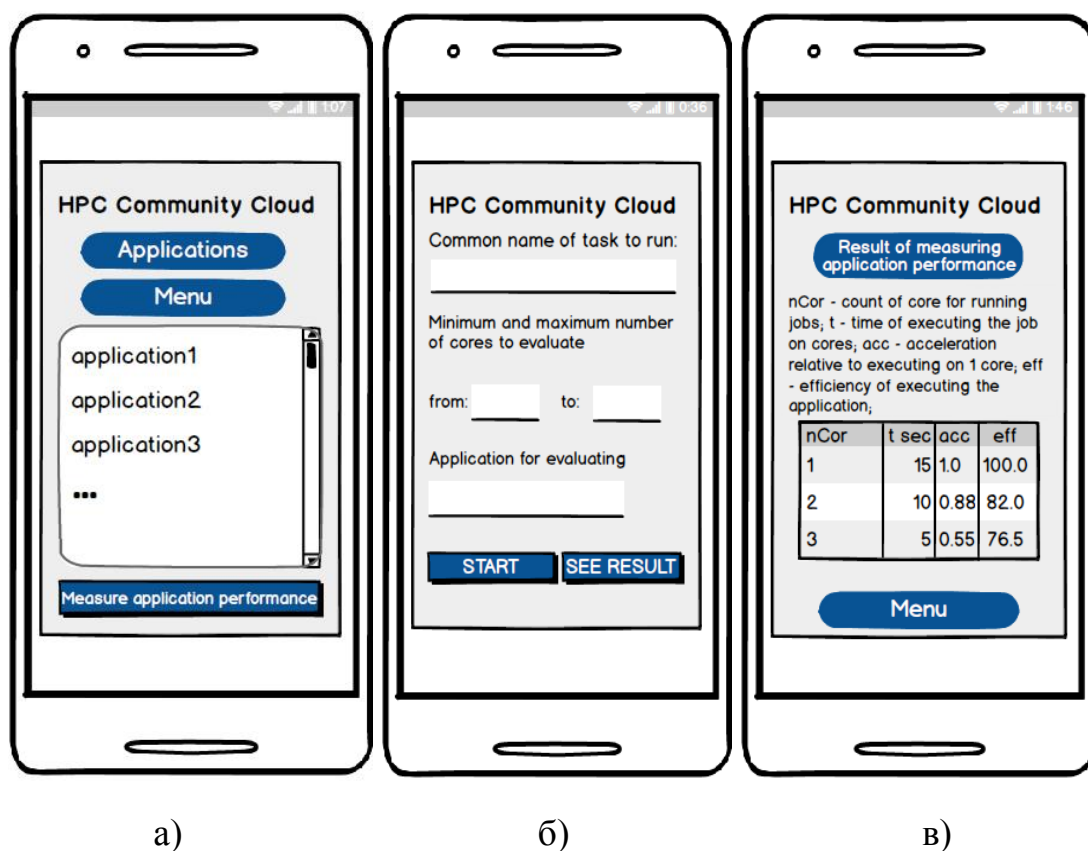


Рисунок 19 – Окна списка доступных приложений (а), задание параметров профилировщика (б) и просмотра результатов (в)

2.3. Инструменты для генерации специальных интерфейсов под конкретное вычислительное приложение

2.3.1 Описание конструктора типов и их спецификации

Во время разработки сценариев в виде вычислительных моделей пользователю требуется специфицировать созданные переменные определённым образом для последующей генерации интерфейсов входных данных вычислительного приложения. То есть, помимо имени и координат в окне пользователь может дополнительно определить такие параметры как, например, значение переменной, если оно статично и другие.

Однако не всегда созданная переменная проста по своему составу, часто встречаются случаи, когда пользователю требуется ввести значение некоторой переменной, которая состоит из числа фиксированных простых полей и множества произвольных объектов, причём эти объекты также могут быть разными по своей природе. Такие переменные необходимо как-то описывать и отличать от более простых случаев, чтобы впоследствии генерировать их особым образом, поэтому в рамках магистерской диссертации в качестве такого описания предлагается ввести в контексте вычислительных моделей типизацию переменных, наподобие типизации в высокоуровневых языках и хранить описание всех созданных типов в некоторой базе типов.

Проектом предполагается, что система, различая эти типы и находя их описание в базе, по данному описанию будет предоставлять пользовательский интерфейс для конкретной спецификации значений данных величин. Каждая переменная вычислительной модели подразумевается как некоторый новый тип, относящийся к одному из доступных и понятных системе классов типов, причём пользователь имеет возможность при создании новой переменной указать в качестве её типа один из уже существующих в базе типов.

Использование такой типизации предлагается при создании новой переменной вычислительной модели приложения внутри всплывающего окна для

спецификации параметров созданной переменной. Пользователю предлагается создать для текущей переменной новый тип, используя конструктор типов, разработанный в рамках данной работы, или же выбрать в качестве него один из доступных типов в предложенном ниспадающем списке типов из базы типов.

Конструктор типов позволяет специфицировать новый тип, согласно одному из существующих классов типов, описанных ниже (рисунок 20). Он предоставляет пользователю выбрать класс будущего типа из общего списка доступных классов, после чего на основе выбора пользователю будет предоставлен соответствующий для данного класса конструктор. После спецификации типа полученные данные будут сохранены в виде схемы типа в общую базу типов как новый тип с именем переменной, что в дальнейшем позволит на основе сохранённой информации визуализировать данную переменную уже для конкретной спецификации перед запуском задачи непосредственно.

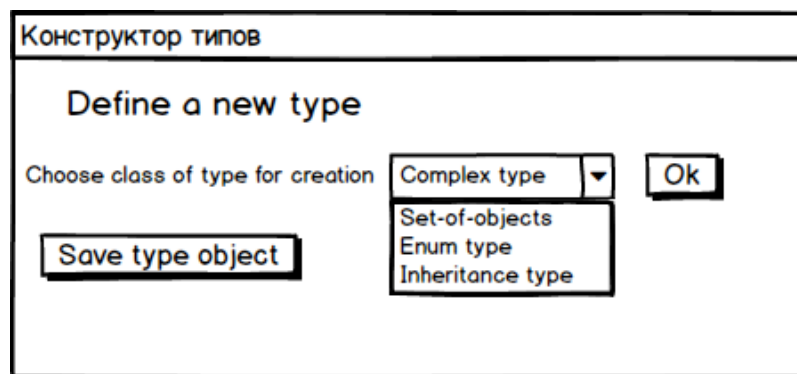


Рисунок 20 – Макет конструктора типов

2.3.2. Классификация типов данных

Проект предполагает, что система должна различать следующие классы типов:

- класс “Complex type” – класс сложного типа;
- класс “Set-of-objects type” – класс множества объектов;
- класс “Enum type” – класс перечисления значений;
- класс “Inheritance type” – класс-наследник;
- класс “Base type” – класс базовых типов.

Класс “*Base type*”, класс сложных типов – данный класс объединяет в себе основные типы, распространённые в различных высокоуровневых языках, например, целочисленный тип, строковый, логический, вещественный и так далее, данный класс не имеет возможности расширяться посредством добавления новых типов, все типы, входящие в данный класс добавлены в систему при создании и являются простейшими составляющими более сложных типов.

Класс “*Complex type*”, класс сложных типов – данный класс предназначен для сложных объектов данных, которые могут состоять из некоторого потенциально бесконечного числа атрибутов, на практике ограниченного реализацией, причём каждый атрибут в свою очередь, помимо имени атрибута имеет ещё и тип атрибута, в качестве которого может выступать любой из созданных ранее пользователем тип или один из заложенных при создании в систему базовых типов (рисунок 21).

Рассмотрим пример данного типа: тип Круг, у которого атрибутами могут выступать радиус (являющийся числом), позиция на экране по вертикали и по горизонтали (являются числами), цвет круга (набор нескольких значений, предоставляющих пользователю выбор) и так далее.

The image shows a software window titled "Конструктор типов" (Type Constructor). Inside, the main heading is "Define a new type". Below this, there is a section "Choose class of type for creation" with a dropdown menu currently showing "Complex type" and an "Ok" button. To the left of the main area is a "Save type object" button. On the right, under the heading "New complex object", there is a text input field for "Name of complex object:". Below this are two buttons: "Add type elem" and "Delete type elem". Further down are two more input fields labeled "Name:" and "Type:". At the very bottom of the dialog, there are three dots "...".

Рисунок 21 – Макет создания типа “Complex type”

Класс “*Set-of-objects type*”, класс множества объектов – данный класс предназначен, как исходит из названия, для множества объектов, объединённых в один общий объект и предоставляет пользователю выбор внутреннего объек-

та (элемента). Внутри объекта данного класса в любой момент времени содержится известное количество элементов, причём эти элементы могут являться объектами, созданными ранее пользователем и хранящимися в базе (рисунок 22), либо же создаваться при определении типа, принадлежащего к данному классу и в дальнейшем рассматриваться системой как отдельные объекты, которые можно использовать вне данного типа (рисунок 23).

Примером данного типа является тип Фигуры, в состав которого могут входить сложные типы, такие как квадрат, круг, треугольник и так далее, причём некоторые внутренние типы могут быть определены при спецификации данного типа, например, дополнительно к имеющимся фигурам можно добавить ещё несколько. Кроме сложных типов в составе данного множества объектов могут присутствовать и другие типы, например общий цвет фигур, который может принимать некоторое конкретное значение или же являться набором значений, предоставляющих пользователю выбор.

Конструктор типов

Define a new type

Choose class of type for creation: Set-of-objects type [v] [Ok]

[Save type object]

Added a new set-of-objects type

Name of set-of-objects type: []

☒ Set the name of exist types ☐ Define new type

Enter existing type names with a comma

[]

Рисунок 22 – Макет создания типа “Set-of-objects type”, где перечисляются внутренние объекты

The dialog box is titled "Конструктор типов" (Type Constructor). It contains the following elements:

- Title:** Define a new type
- Choose class of type for creation:** A dropdown menu showing "Set-of-objects type" and an "Ok" button.
- Buttons:** "Save type object" (highlighted), "New type object", and "Delete type object".
- Text:** "Added a new set-of-objects type"
- Form:** "Name of set-of-objects type:" followed by a text input field.
- Radio Buttons:** "Set the name of exist types" (unselected) and "Define new type" (selected).
- Form:** "New type object" section with "Name of type of object:" followed by a text input field.
- Buttons:** "Add type elem" and "Delete type elem".
- Form:** "Name:" followed by a text input field.
- Form:** "Type:" followed by a text input field.
- Footer:** Three dots "..."

Рисунок 23 – Макет создания типа “Set-of-objects type”, где определяется новый внутренний объект

Класс “*Enum type*”, класс перечисления значений – данный класс позволяет описывать типы, представляющие собой так называемые перечисления, когда необходимое значение может изменяться в зависимости от ситуации, но его суть остаётся неизменной. При определении типа такого класса пользователем заносятся доступные для выбора значения в систему непосредственно (рисунок 24).

Для данного класса примером типа является тип Цвет, значениями которого могут быть несколько различных оттенков, например, красный, зелёный и так далее.

The dialog box is titled "Конструктор типов" (Type Constructor). It contains the following elements:

- Title:** Define a new type
- Choose class of type for creation:** A dropdown menu showing "Enum type" and an "Ok" button.
- Buttons:** "Save type object" (highlighted), "Add enum value", and "Delete enum value".
- Text:** "Added a new enum type"
- Form:** "Name of enum type:" followed by a text input field.
- Form:** "Value:" followed by a text input field.
- Form:** "Value:" followed by a text input field.
- Footer:** Three dots "..."

Рисунок 24 – Макет создания типа “Enum type”

Класс “*Inheritance type*”, класс-наследник – данный класс позволяет определить тип, который полностью наследует характеристики своего так называемого родителя, одного или нескольких, но при этом в дополнение к ним имеет собственные атрибуты, которые, как и в случае с классом “*Complex type*”, имеют имя, а также тип, который может являться одним из находящихся в базе типов (рисунок 25). В случае данного класса мы имеем строгую привязку класса-наследника к родительским типам, которая выражается в том, что при удалении одного или нескольких родительских типов из базы, класс-наследник автоматически перестаёт существовать и подлежит удалению, при этом удаление класса-наследника никак не повлияет на родительские типы.

В качестве примера типа из данного класса можно рассмотреть тип Студент, наследующий от типа Человек некоторые атрибуты, например, возраст, имя, адрес и так далее. Однако помимо наследуемых атрибутов тип имеет и собственные, к примеру, номер зачётной книжки, шифр студента и другие.

The image shows a software window titled "Конструктор типов" (Type Constructor). Inside, the main heading is "Define a new type". Below this, there is a label "Choose class of type for creation" followed by a dropdown menu currently set to "Inheritance type" and an "Ok" button. To the left of the main area is a "Save type object" button. On the right, a message states "Added a new inheritance type". Below this message are two input fields: "Name of inheritance type:" and "Inherited types:". Further down are two buttons, "Added attribute" and "Delete attribute". Below these are two more input fields labeled "Name:" and "Type:". At the bottom center, there are three dots "...".

Рисунок 25 – Макет создания типа “Inheritance type”

2.3.3 Визуализация типов данных

Визуализация входных данных вычислительного приложения происходит после сохранения некоторого множества входных и выходных величин внутри вычислительной модели, как так называемую проблему, которую впоследствии специфицировав конкретными значениями отправляют на вычисление на су-

персональном компьютере с помощью сервиса HPC2C и вспомогательных внутренних сценариев, оформленных в виде вычислительных моделей. Визуализация входного интерфейса пользователя происходит согласно классификации выбранных типов.

При визуализации типа, принадлежащего к классу “Base type”, классу базовых типов, будут отображены простые формы, в зависимости от выбранного для визуализации типа.

При визуализации типа из класса “Complex type”, класс сложного типа, будут отображены все определённые пользователем при создании атрибуты, согласно их собственным типам, указанным при определении.

При визуализации типа, относящегося к классу “Set-of-objects type”, классу множества объектов, пользователю будет предоставлен выпадающий список доступных для отображения элементов, в котором он сможет выбрать необходимый для своих целей внутренний элемент и отобразить его необходимое количество раз или же удалить его за ненадобностью.

При визуализации типа, который относится к классу “Enum type”, классу перечисления значений, пользователю отобразится выпадающий список возможных значений данного типа и предоставится возможность выбора интересующего его значения из данного списка.

При визуализации типа, принадлежащего к классу “Inheritance type”, классу-наследнику, будут отображены сначала атрибуты всех типов, определённых для данного типа в качестве родительских типов, после чего отобразятся и собственные атрибуты визуализируемого типа.

2.3.4. Валидация объектов данных

Перед отправкой вычислительной задачи на выполнение на вычислительную систему в предварительно сгенерированном интерфейсе для входных параметров пользователю необходимо задать значения входным переменным, после чего система перед отправкой их вместе с алгоритмом приложения на сер-

вер для обработки и запуска обязана удостовериться, что отправляемые данные из переменных соответствуют указанным для каждой из переменных в вычислительной модели типам, то есть валидны, и, в случае провала проверки, сообщить пользователю о некорректном вводе данных.

Для проверки валидности переменной помимо схемы типа, хранящейся в базе типов также необходима вычислительная модель валидатор, которая позволит проверить, что те данные, которые были поданы на вход действительно относятся к проверяемому типу. Данная вычислительная модель при проверке на валидность структуры с одной стороны может использовать только схему типа из базы, а с другой стороны может содержать какие-либо дополнительные проверки, которые в зависимости от проверяемого типа могут различаться. Использование вычислительной модели валидатора целесообразно для нетривиальных типов, имеющих сложную структуру, нет необходимости в проверке на валидность с помощью вычислительной модели типов, принадлежащих к классу “Base type”, так как для данных типов можно реализовать валидатор проще.

Однако, существует проблема, что кто-то мог удалить, например файлы, на которые ссылается некоторый объект данных случайно или же намеренно, в таком случае объект, ссылающийся на удалённые файлы становится неактуальным. Следовательно, системе необходимо каким-то образом проверять, что все ссылки некоторого объекта данных актуальны и файлы, связанные с этим объектом не удалены, а при отсутствии таких файлов удалять объект данных или уведомлять пользователя об отсутствии искомых файлов, помимо проверки на валидность структуры объекта.

Кроме того, остаётся фундаментальная проблема восстановления семантики из текста: не всегда можно формальным способом определить, что рассматриваемый объект соответствует определенному смыслу эта задача может оказаться алгоритмически неразрешимой. Поэтому при проверке некоторого объекта данных на то, что он является или не является тем самым, что имел в виду пользователь, можно проверить какие-то признаки, указывающие на вложен-

ный пользователем смысл, но не всегда можно утверждать на основе формальных проверок, что объект соответствует тому, что ожидается на вход некоторой вычислительной операции. В рамках магистерской диссертации предлагается ограничить проверку на валидность объекта данных проверкой соответствия структуры объекта схеме типа из базы типов, который указывался при создании объекта, с помощью вычислительной модели-валидатора, при необходимости дополнив её произвольными проверками.

2.3.5. Управление данными

Проблема управления данными относительно разрабатываемого сервиса заключается в следующем: доступ к различным объектам данных, моделям, задачам и так далее должен быть только у авторизованного конкретного пользователя, их не должны видеть все пользователи, авторизованные в сервисе НРС2С. Кроме того, необходима возможность разделения между пользователями доступа к объектам для возможной совместной работы над ними, причём пользователь может предоставить доступ как некоторой группе, так и отдельному пользователю, оба варианта необходимо различать внутри системы.

Помимо общего доступа к файлам конкретного пользователя существует проблема многократного копирования данных между несколькими устройствами. Данные подготавливаются в некотором каталоге, перед запуском задачи на НРС2С данные отправляются на вычислительную систему, используются и результат вычисления пересылается обратно. При многократном запуске одного и того же приложения возникает множество копий таких данных, но если некоторые из них могут быть именованы как объект данных, то другие могут оказаться промежуточными данными, которые следует удалить после получения требуемых результатов.

В качестве решения проблемы многократного копирования данных предлагается при описании некоторого объекта данных, который хранит в себе информацию о местоположении требуемых файлов также указывать и местопо-

ложение копий этих данных, если они имеются. Следовательно, если системе требуются определённые данные на каком-либо устройстве, она может опросить данный файл и убедиться в наличии или же отсутствии на данном устройстве копии требуемых данных и, в зависимости от результата, скопировать в систему данные, либо воспользоваться имеющимися.

2.4. Usability

Usability (с англ. – “удобство использования”) – основной критерий оценки интерфейса, определяющий насколько просто и удобно использовать то или иной интерфейс. Данный критерий в основном измеряется наличием или же отсутствием проблем у пользователя при использовании некоторого интерфейса. Выделяется несколько качественных критериев юзабельности продукта [23]:

1. Эффективность – насколько быстро и просто пользователь выполняет требуемые действия для достижения поставленной цели. Данный критерий можно измерить по количеству реализуемых задач, отношению числа успешных действий к ошибкам и количеству используемых функций.
2. Продуктивность – оценка объёма ресурсов, требуемых для точного решения поставленных пользователем задач. Можно характеризовать временем на обучение и выполнение задания, количеством совершаемых ошибок, временем, затрачиваемым на их решение и так далее.
3. Удовлетворённость – насколько пользователь доволен используемым интерфейсом. Данный критерий сложноопределяем, поскольку является субъективным и на формирование того или иного мнения влияет множество факторов. Чаще всего измеряется при помощи рейтинговой оценки по шкалам полезности продукта, удовлетворённости функционалом и другим показателям.

На основе вышеперечисленных критериев, проанализировав интерфейс, можно сделать вывод о том насколько юзабелен разработанный интерфейс и что нужно сделать для повышения его юзабельности. Применимо к данной работе можно сделать вывод, что помимо генерации пользовательских интерфейсов возникает вопрос как данные интерфейсы сделать удобными и простыми в использовании с точки зрения юзабельности и критериев качества.

Существует несколько способов проверки юзабилити интерфейсов, рассмотрим несколько из них. Наиболее простым методом оценки является анализ статистики. Для его реализации достаточно воспользоваться готовыми средствами, например Яндекс.Метрика или GoogleAnalytics, позволяющие оценить посещаемость, а также проанализировать поведение пользователя внутри интерфейса. Однако данные выводы могут быть довольно поверхностными и их может оказаться недостаточно для определения юзабельности.

Другой способ проверки на юзабельность – организовать обратную связь. Пользователям предоставляется возможность оставить отзыв и на основе полученных отзывов реализовывать соответствующие улучшения интерфейса. Однако в большинстве случаев такой способ позволяет выявить наиболее яркие недостатки интерфейса, а также существует вероятность того, что пользователь откажет в составлении отзыва или же внесёт недостоверную информацию.

Тестирование страниц или А/В-тестирование позволяет сравнивать различные версии интерфейсов или оценить нововведения на специально сформированной фокус-группе. Данную группу делят на две части и каждая работает с определённым видом интерфейса, например одна часть исследует уже готовый интерфейс, а вторая тестирует нововведения внутри готового интерфейса. Наблюдая за действиями пользователей, например, с помощью тепловых карт, показывающих частоту использования мыши в определённых местах, а также используя оценку пользователей, можно сделать более полный вывод об удобности использования того или иного интерфейса.

Спроектированные и разработанные базовые средства согласно поставленным задачам позволяют генерировать пользовательские интерфейсы на основе спецификации входных и выходных данных вычислительных экспериментов, однако то, в какой виде должны генерироваться такие интерфейсы с точки зрения usability это сложный и трудоёмкий вопрос, требующий дальнейших исследований для определения наиболее удобного вида генерируемого интерфейса. Тем не менее, проанализировав несколько сгенерированных интерфейсов на начальных этапах разработки были сделаны улучшения отображения компонентов интерфейса с точки зрения usability.

Например, при использовании конструктора типов было принято решение вынести данный модуль в отдельную вкладку, а не создавать дополнительное модальное окно, что позволило разгрузить область работы с переменными и избежать путаницы в модальных окнах. При генерации интерфейса для входных величин можно сгенерировать все поля, соответствующие переменным, подряд, однако если переменных много, то подобное отображение становится нечитабельным из-за большого количества элементов идущих подряд, а также становится сложно разграничивать такие элементы. Для избежания подобной ситуации было принято решение отображать каждую переменную в отдельной сворачиваемой вкладке, внутри которой пользователю будет предоставлена возможность заполнения параметров элемента.

2.5. Визуализация результатов вычислительного приложения

Визуализация выходных данных вычислительного приложения не менее важная часть, чем генерация интерфейсов входных данных для конечного пользователя. Вариантов визуализации данных довольно много, самый простой из них – выводить пользователю непосредственно результат работы приложения, однако данный способ не всегда удобен для пользователя с точки зрения usability. Например, результатом работы вычислительного приложения является огромный двумерный массив, и передав его пользователю в качестве выходных

данных, есть вероятность того, что пользователь, не знакомый с алгоритмом работы приложения сочтёт данное представление информации не информативным. А вот представление того же массива в виде таблицы или графика с соответствующей легендой будет уже куда удобнее анализировать конечному пользователю.

Другой вариант визуализации выходных данных – визуализировать результат работы приложения в тех средствах, в которых были заданы входные величины. То есть предоставить пользователю выходные данные в таких же формах как и введённые перед запуском приложения входные данные. Данный способ удобен, если типы начальных и результирующих данных совпадают или являются простыми для анализа без дополнительных манипуляций.

Однако и у данного способа имеются свои недостатки, к примеру, рассмотрим случай, когда на вход приложению подаётся какое-то изображение и на выход пользователь также получает изображение. В таком случае, если пользователь задавал входное изображение как ссылку на некоторый файл или путь до данного файла в системе, то для выходного изображения такое представление не всегда удобно. Часто пользователю удобнее анализировать изображение сразу после получения и не тратить время на поиск его в некоторой системе. Тем не менее данный способ намного удобнее, чем программное отображение изображения, которое может быть непонятно пользователю.

Когда же входные и выходные величины различны по своему типу способы визуализации таких выходных величин могут быть индивидуальны и зависеть от конкретной реализации вычислительного приложения. Следовательно, некоторый общий подход визуализации результатов расчётов вычислительных задач, рассматриваемых нами, проследить можно, но он покрывает сравнительно небольшое множество реализаций визуализаторов выходных данных.

В рамках магистерской диссертации предлагается следующий подход: пользователю при описании нового типа интересующего его предлагается написать некоторый дополнительный плагин, связанный с данным типом,

например, с помощью средств HTML и JavaScript, позволяющий визуализировать данный тип необходимым пользователю образом. Такой плагин сохраняется в системе и позволяет использовать себя при визуализации результатов согласно тому типу, к которому он привязан, причём таких плагинов может быть несколько.

Например, результатом вычислений является трёхмерный массив, тогда один из возможных плагинов может визуализировать его как статичный куб, другой – как куб, который можно вращать, перемещать, отдалять и т. д. Если же для некоторого типа отсутствуют плагины для его визуализации, то пользователю предлагается просмотреть результат выполнения задачи в том виде, в котором они были получены после выполнения расчётов. Таким образом каждый объект, имеющий тип, помимо него может иметь счётное множество его представлений, определённое пользователем.

3. РЕАЛИЗАЦИЯ СИСТЕМЫ

3.1. Выбор средств для реализации программного комплекса

В рамках магистерской диссертации были использованы следующие средства для реализации заявленных при постановке задачи компонентов: поскольку основная функциональность сервиса HPC Community Cloud (HPC2C) и интерфейсной системы для работы с вычислительными моделями Aka реализована с помощью языка разметки HTML, языка JavaScript (JS) и языка описания CSS, то конструктор типов, а также визуализатор сконструированных типов были реализованы аналогичным способом для удобства встраивания в данный комплекс и взаимодействия с сервисами. Для взаимодействия с API HPC2C используется XMLHttpRequest – API, позволяющий отправлять HTTP и HTTPS запросы напрямую к некоторому веб-серверу и загружать данные ответа сервера напрямую в вызываемый скрипт.

Мобильный интерфейс для взаимодействия с HPC2C был реализован на платформе Android с помощью Android Studio – интегрированной среды разработки для работы с платформой Android. Языком разработки интерфейса является Java. Выбор данных средств обусловлен бесплатным доступом к среде разработки, большим количеством tutorиалов по разработке приложений с помощью среды Android Studio, простотой отладки приложения на мобильном устройстве с операционной системой Android и опытом работы с языком Java. Для взаимодействия мобильного интерфейса с API HPC2C используется библиотека Volley, преимуществами которой является поддержка нескольких одновременных сетевых подключений, поддержка приоритезации запросов, возможность отмены запроса, простота настройки и другие.

3.2. Описание модельного приложения

В качестве модельного приложения для демонстрации корректности работы компонентов программного продукта была выбрана модель Фриша-

Хаслахера-Помо – Multi-Particle (FHP-MP) [24-26], принадлежащая классу клеточных автоматов с решётчатым газом и разработанная в институте вычислительной математики и математической геофизики Сибирского отделения Российской академии наук (ИВМиМГ СО РАН). Данная модель позволяет моделировать течение газа в трубе с препятствиями внутри, например, в форме прямой линии или прямоугольника (рисунок 26).

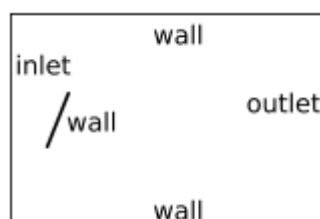


Рисунок 26 – Первоначальное глобальное состояние, где inlet – входные ячейки, outlet – выходные ячейки, а wall – настенные ячейки; препятствие в форме линии

В ИВМиМГ разработан программный комплекс, реализующий модель FHP-MP. Комплексом предусматривается три основных действия для исследователя модели FHP-MP:

- построение первоначального глобального состояния клеточного автомата;
- моделирование интересующих состояний клеточного автомата;
- постобработка для получения результата моделирования.

Эти действия реализуются тремя отдельными программами: `fhpmake`, используется для построения первоначального состояния клеточного автомата, `fhpsimulation` – приложение для запуска симуляции на основе некоторого состояния и `fhpvizualization` – постобработка состояний с предыдущего шага.

Все три приложения требуют предоставления файлов конфигурации с параметрами, требуемыми для корректного выполнения приложений в качестве входных данных. Это означает, что каждый раз, когда необходимо создать новое глобальное состояние клеточного автомата в качестве начальных условий

для моделирования с помощью приложения `fhpmake`, запустить симуляцию через `fhpsimulation` или визуализацию с помощью `fhpvizualization`, пользователю требуется выбрать ранее подготовленные файлы конфигурации или же создать новые.

Симуляция может начинаться с файла глобального состояния клеточного автомата, подготовленного с помощью `fhpmake`, или продолжаться с определённого глобального состояния, полученного во время предыдущих симуляций. Симуляция создаёт серию файлов с глобальными состояниями, которые соответствуют определённым итерациям симуляции по запросу пользователя в файле конфигурации `fhpsimulation`. Пользователь имеет возможность продолжить ранее выполненное моделирование с большим количеством итераций или выбрать одно из имеющихся состояний в каталоге моделирования и повторить вычисления с теми же или изменёнными параметрами, что будет восприниматься системой как новый эксперимент.

Постобработка `fhpvizualization` вызывается для определённого глобального состояния или для ряда глобальных состояний, полученных на этапе симуляции или построения начального состояния клеточного автомата. Данное приложение позволяет получить растровые изображения, соответствующие файлам состояний, а также файлы с числовыми данными для усреднённых полей скорости и концентрации для каждого глобального состояния.

3.3. Демонстрация работы модельного приложения внутри системы НРС2С

3.3.1. Общий вид разрабатываемого вычислительного сценария

Для описанного ранее (см. раздел 3.2) модельного приложения клеточного-автоматного моделирования был сконструирован вычислительный сценарий, см. Приложение Г, согласно трём основным этапам моделирования, описанным выше. Каждый этап имеет возможность, помимо последовательного выполне-

ния этапов друг за другом, выполнения этапа вне зависимости от двух других, при условии наличия требуемых входных данных. Для наглядности сценария некоторые операции, подразумевающие большой набор действий, прописаны довольно крупно, можно сказать скрыты за некоторым обозначением, а логика таких операций вынесена в отдельные сценарии, на которые происходит ссылка внутри главного сценария, причём часть таких внутренних сценариев может быть написана пользователем.

Другая же часть внутренних сценариев является системной и позволяет выполнять некоторые стандартные действия по отношению к HPC Community Cloud (HPC2C), например отправка задачи на выполнение и другие, либо же часто используемые действия пользователя, которые были проанализированы и на их основе сконструированы сценарии. Системные сценарии в общем списке существующих сценариев имеют приписку Ака, что позволяет отличать их от пользовательских.

Для каждого из трёх этапов моделирования требуются файлы конфигурации с параметрами, помимо некоторых других данных, требуемых на вход, поэтому примеры таких файлов были проанализированы, выявлена их структура и виды параметров для корректного внесения их в вычислительный сценарий и продумана логика отображения вычислительных операций, соответствующих этапам моделирования, кроме того для последнего этапа постобработки был реализован плагин, отображающий результаты обработки данных в виде карусели с изображениями, полученными на последнем этапе моделирования.

В качестве наглядного примера рассмотрим первый этап получения первоначального состояния клеточного автомата и присущие этому этапу шаги, начиная с планирования модели и заканчивая вычислением. Для остальных этапов моделирования рассматриваемые ниже действия являются аналогичными, за исключением смысла входных/выходных данных и логики алгоритма.

3.3.2. Проектирование вычислительного сценария для получения начального состояния клеточного автомата

При анализе конфигулятора `fhpmake.conf`, см. Приложение Д, подающего на вход приложению `fhpmake` для получения файла глобального состояния клеточного автомата, были выявлены особенности строения файла. Во-первых, данный файл реализован в формате `ini`, во-вторых, все параметры конфигулятора объединены в соответствующие смыслу файла группы:

- `main` – группа параметров приложения, представляющая главную секцию, например, тип стенок, имя выходного файла и другие; на момент исследования главная секция являлась прямоугольником;
- `circle` – группа параметров, соответствующая препятствию в трубе в виде круга, например, позиция внутри главной секции, тип пролетающих частиц и так далее;
- `bar` – параметры, принадлежащие препятствию в виде прямоугольника или квадрата, такие как координаты вершин, концентрация газа и прочие;
- `line` – параметры, описывающие препятствие в виде линии, например начальные и конечные координаты.

При рассмотрении примера конфигурационного файла можно сделать вывод, что секции с препятствиями различных видов, в отличие от главной секции могут быть использованы несколько раз, так как препятствий может быть много, следовательно, имеет смысл при создании модели вынести описание препятствий в некоторую отдельную процедуру или в отдельный сценарий и сослаться на него в главном сценарии.

Таким образом, в сценарии создания начального состояния (рисунок 27), являющемся частью общей модели для моделирования состояний клеточного автомата, мы определяем переменные главной секции и общий вид препятствий для последующей спецификации всех объектов и генерации пользовательского интерфейса по предложенной спецификации.

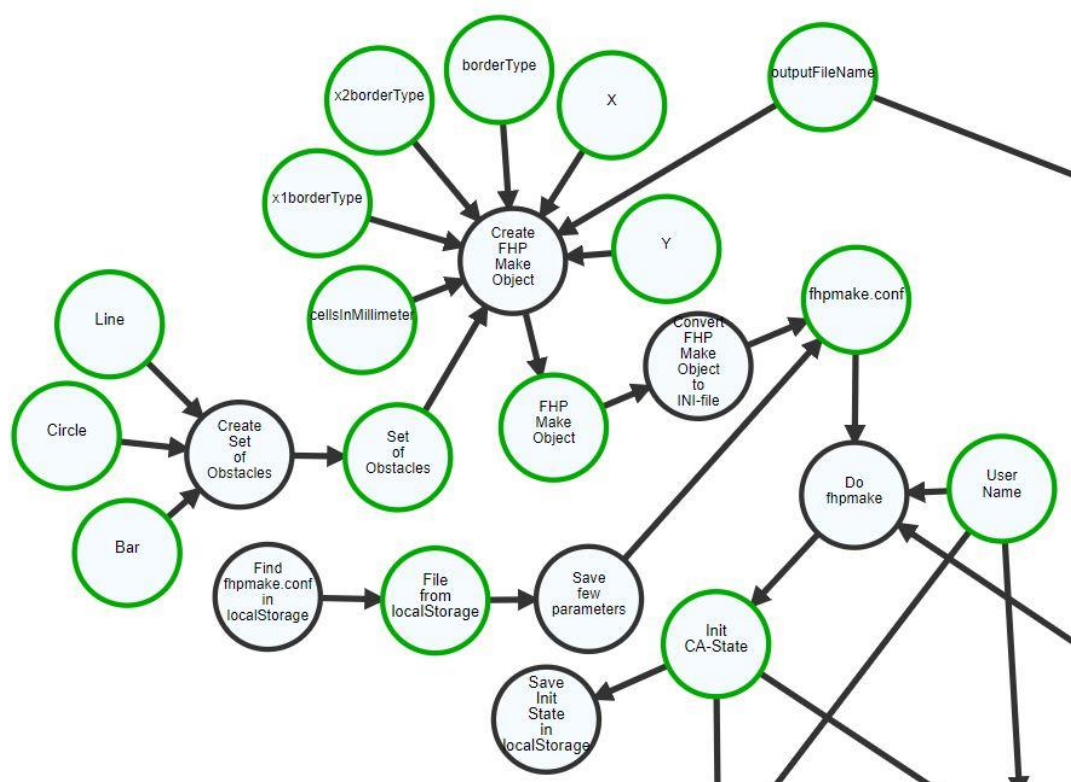




Рисунок 27 – Вычислительный сценарий, позволяющий создать первоначальное глобальное состояние клеточного автомата

После добавления требуемых переменных в сценарий для каждой из них требуется описать тип переменной, на основе которого в дальнейшем будут генерироваться интерфейсы и осуществляться проверка введённых пользователем значений на валидность. Для присваивания переменной типа, необходимо в модальном окне переменной (рисунок 28) выбрать кнопку Set a type for value , которая добавит поле в модальном окне и предложит пользователю определить тип переменной одним из доступных способов: прописать структуру типа в виде JSON-схемы в предложенном одноимённом поле или воспользоваться конструктором типов, нажав на кнопку Create type in Types Creator  (рисунок 28) и описать тип с помощью конструктора.

variable

title	borderType
x	258
y	-334
input	false

JSON-Schema

```
{
  "$schema": "http://json-schema.org/schema#",
  "title": "borderType",
  "type": "string",
  "enum": [
    "wall",
    "inlet",
    "outlet",
    "conventional"
  ]
}
```

+ ≡ + ⊕

OK

Рисунок 28 – Модальное окно переменной с указанным типом

После выбора определения типа через конструктор типов пользователь переходит в окно с конструктором (рисунок 29), где ему предлагается выбрать класс создаваемого типа (рисунок 29-31) и заполнив предложенные поля нажать кнопку Save type of object (рисунок 29) для сохранения структуры типа в базу. При сохранении типа в базу структура типа также отобразится в поле JSON-Schema модального окна переменной (рисунок 28), созданного в начале.

Define a new type for variable

Chose class of type for creation Complex type Ok

Save type of object

Type of object

Name of type of object:

Add type elem Delete type elem

Name:

Type:

Рисунок 29 – Создание типа, принадлежащего классу Complex type с помощью конструктора типов

Added a new set-of-objects type

Name of set-of-objects type:

☒ Set the name of exist types ☐ Define new type

Enter existing type names with a comma

Added a new set-of-objects type

Name of set-of-objects type:

☐ Set the name of exist types ☒ Define new type

New type object Delete type object

Type of object

Name of type of object:

Add type elem Delete type elem

Name:

Type:

Рисунок 30 – Создание типа из класса Set-of-objects type в конструкторе типов с помощью перечисления существующих типов (а) и с помощью создания новых внутренних типов (б)

Added a new enum type

Name of enum type:

Add enum value Delete enum value

Value:

Value:

Added a new inheritance type

Name of inheritance type:

Inherited type:

Add attribute Delete attribute

Name:

Type:

Рисунок 31 – Создание типа из класса Enum type (а) и создание типа класса Inheritance type (б) в конструкторе типов

В качестве примера рассмотрим определение типа переменной из главной секции, например, переменная, отвечающая за тип стенок, остальные переменные из секции просты для понимания и описываются простейшими схемами. Переменная Тип стенок представляет собой набор значений, предлагаемых на выбор пользователю, следовательно, переменную можно отнести к классу Enum type (рисунок 32а)). Если рассмотреть более сложную переменную, например переменную, описывающую препятствие круг, то по количеству её полей и различным вариантам конкретизации значений перед постановкой на выполнение можно отнести данную переменную к классу Complex type (рисунок 32б), тип cellsType аналогичен по своей структуре типу borderType). А если рассматривать все препятствия в целом как некоторую переменную Set of Obstacles, то она удовлетворяет условиям класса Set-of-objects, причём определить её тип можно, указав названия внутренних элементов, после определения

внутренних объектов или же создавая их в процессе создания самого Set of Obstacles.

The interface is divided into two main sections for defining variable types:

- Added a new enum type (Left Panel):**
 - Name of enum type:** borderType
 - Buttons:** Add enum value, Delete enum value
 - Values:**
 - Value: wall
 - Value: inlet
 - Value: outlet
 - Value: conventional
- Type of object (Right Panel):**
 - Name of type of object:** Circle
 - Buttons:** Add type elem, Delete type elem
 - Defined Variables:**
 - Name:** x, **Type:** integer
 - Name:** y, **Type:** integer
 - Name:** r, **Type:** integer
 - Name:** cellsType, **Type:** cellsType
 - Name:** gasConcentration, **Type:** integer
 - Name:** gasRestConcentration, **Type:** integer

Рисунок 32 – Определение типа переменных borderType (а) и Circle (б)

3.3.3. Визуализация входных данных для моделирования клеточного автомата

После присваивания типов переменным и сохранения их в базу, для генерации интерфейса входных данных требуется объявить в модели входные и выходные переменные, далее в правой половине окна, где находятся информационные характеристики модели и запущенных задач, в поле Problem name необходимо указать имя будущей задачи, сохранить модель в базе и нажать на кнопку Save as a problem (рисунок 14) для постановки новой задачи. Для корректной постановки задачи требуется в качестве входных величин выбрать все величины, принадлежащие главной секции и величину Set of Obstacles, которая позволит задать требуемое количество входных препятствий того или иного типа (рисунок 33).

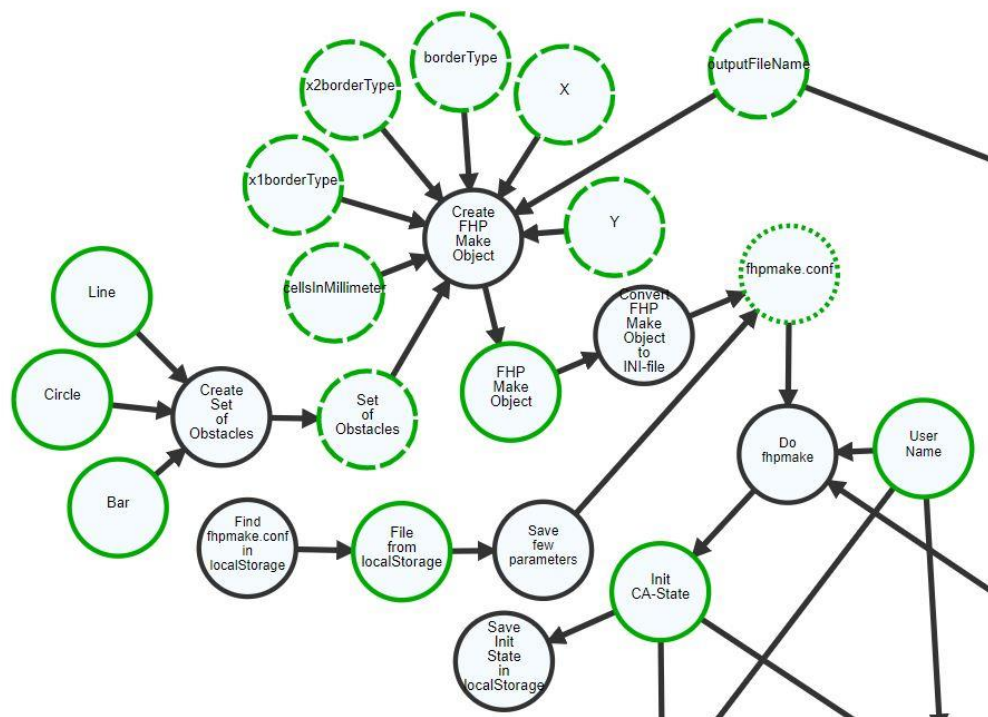


Рисунок 33 – Выбор входных и выходных переменных для создания пользовательского интерфейса

Для просмотра сгенерированного интерфейса и определения величин перед отправкой задачи на выполнение необходимо перейти на вкладку Задачи и выбрать из списка задач созданную ранее задачу, после чего пользователь автоматически переходит на вкладку Решение задач (рисунок 34), где ему предлагается определить значения величин и задать имя для будущего запуска задачи в области слева. Кнопка Solve Problem отправит специфицированные данные вместе с алгоритмом на выполнение, а кнопка Collapse позволит свернуть все открытые области интерфейса.

Ака: Интерфейсная система для работы с вычислительными моделями

The screenshot shows the Akka interface with the following elements:

- Navigation tabs: Демо-магазин, Конструктор моделей, Задачи, Решение задач (active), Previous Jobs, List of CMs.
- Form section on the left:
 - Set a name for the problem instance: [input field]
 - Please fill-in the form to the right and press: [Solve Problem button]
 - Press to collapse the form: [Collapse button]
- Form section on the right (expanded):
 - X: [input field]
 - Y: [input field]
 - Set of Obstacles: [input field]
 - outputFileName: [input field]
 - cellsInMillimeter: [input field]
 - borderType: [input field]
 - x1borderType: [input field]
 - x2borderType: [input field]

Рисунок 34 – Общий вид сгенерированного интерфейса

Каждый элемент сгенерированного интерфейса находится в отдельной области, которую можно развернуть для ввода значения и после скрыть, что позволяет не нагружать интерфейс и работать только с одной функциональной областью (рисунок 35-36). Кроме того, переменным, тип которых принадлежит к классу Enum type соответствует список из возможных значений данной переменной (рисунок 35б)).

The figure consists of two parts, (a) and (b), showing details of the interface:

а) Shows the expanded form for variables X and Y. Each variable has a header with a collapse icon (X or Y) and an input field below it.

б) Shows the expanded form for the variable x1borderType. It displays a list of possible values: wall, inlet, outlet, and conventional. The 'wall' option is currently selected. Below this, the x2borderType variable is shown with its input field.

Рисунок 35 – Части сгенерированного интерфейса: границы области (а) и типы левой и правой стенки (б)

The image shows a vertical stack of three form sections. Each section has a header with a variable name in blue text and a corresponding input field below it. The first section is for 'outputFileName' and has an empty text input. The second section is for 'cellsInMillimeter' and also has an empty text input. The third section is for 'borderType' and has a dropdown menu with 'wall' selected. The entire form is enclosed in a light gray border.

Рисунок 36 – Части сгенерированного интерфейса: имя выходного файла, тип границ для всех стенок и количество ячеек в миллиметре

Область задания значений переменной Set of Obstacles позволяет определить множество значений одного из трёх доступных типов препятствий, см. Приложение Е, причём внутри препятствий содержатся значения для определения. Кнопка Add Item позволяет добавить новое препятствие, внутри которого в выпадающем списке изменяется тип, а кнопка Delete Item позволяет удалить лишние препятствия.

Визуализация входных переменных в предложенном виде доступна для любого вычислительного сценария, у которого входным переменным был поставлен в соответствие некоторый тип, принадлежащий к одному из доступных классов типов и данное множество переменных вместе с выходными переменными было сохранено в базе как проблема.

3.4. Демонстрация работы мобильного приложения под управлением НРС2С

3.4.1. Основная функциональность приложения

Мобильный интерфейс HPC Community Cloud (HPC2C) предоставляет пользователям доступ к API сервиса. Авторизовавшись в системе (рисунок 37а)) пользователь переходит в меню приложения (рисунок 37б)), где ему предоставляется возможность просмотра интересующих его подпунктов меню (рисунок 38), а также просмотра файловой системы, хранящей файлы, доступные авторизованному пользователю (рисунок 37в)), например файлы созданных приложений, результаты выполненных задач, необходимые документы и так далее.

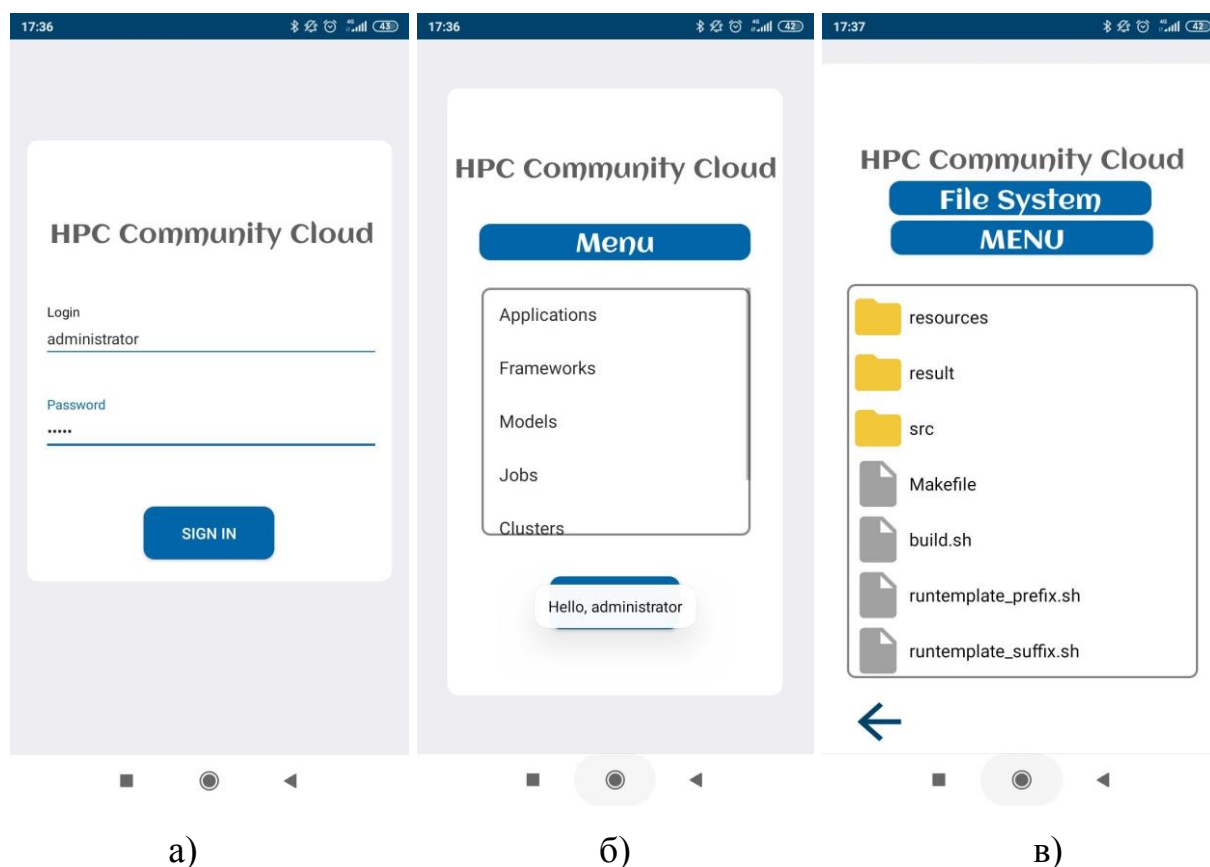


Рисунок 37 – Форма входа в систему (а), меню приложения (б) и файловая система пользователя (в)

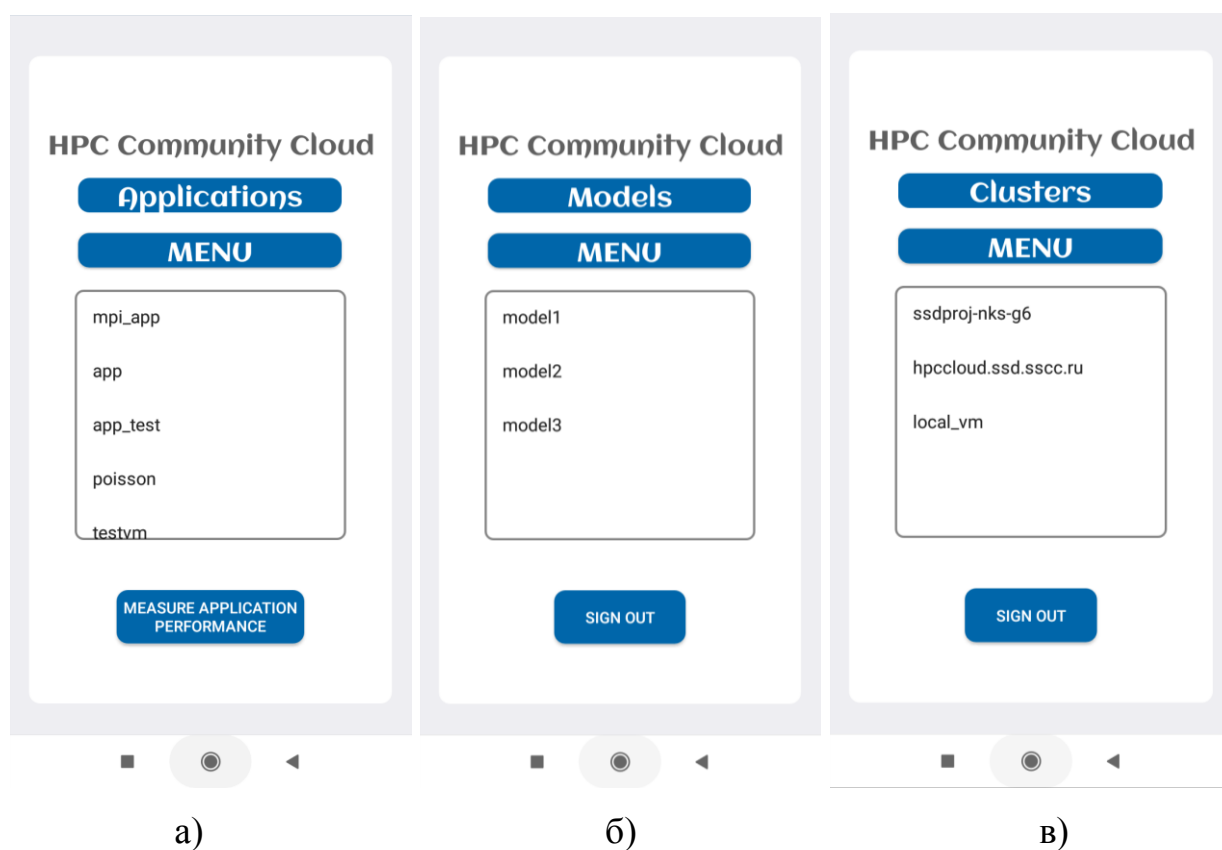


Рисунок 38 – Списки приложений (а), моделей (б) и кластеров (в)

Помимо просмотра имеющихся у пользователя данных, интерфейс предоставляет пользователю возможность постановки новой задачи на выполнение (рисунок 39а)), отслеживание статуса её выполнения (рисунок 39б)) и просмотр результатов вычисления (рисунок 39в).

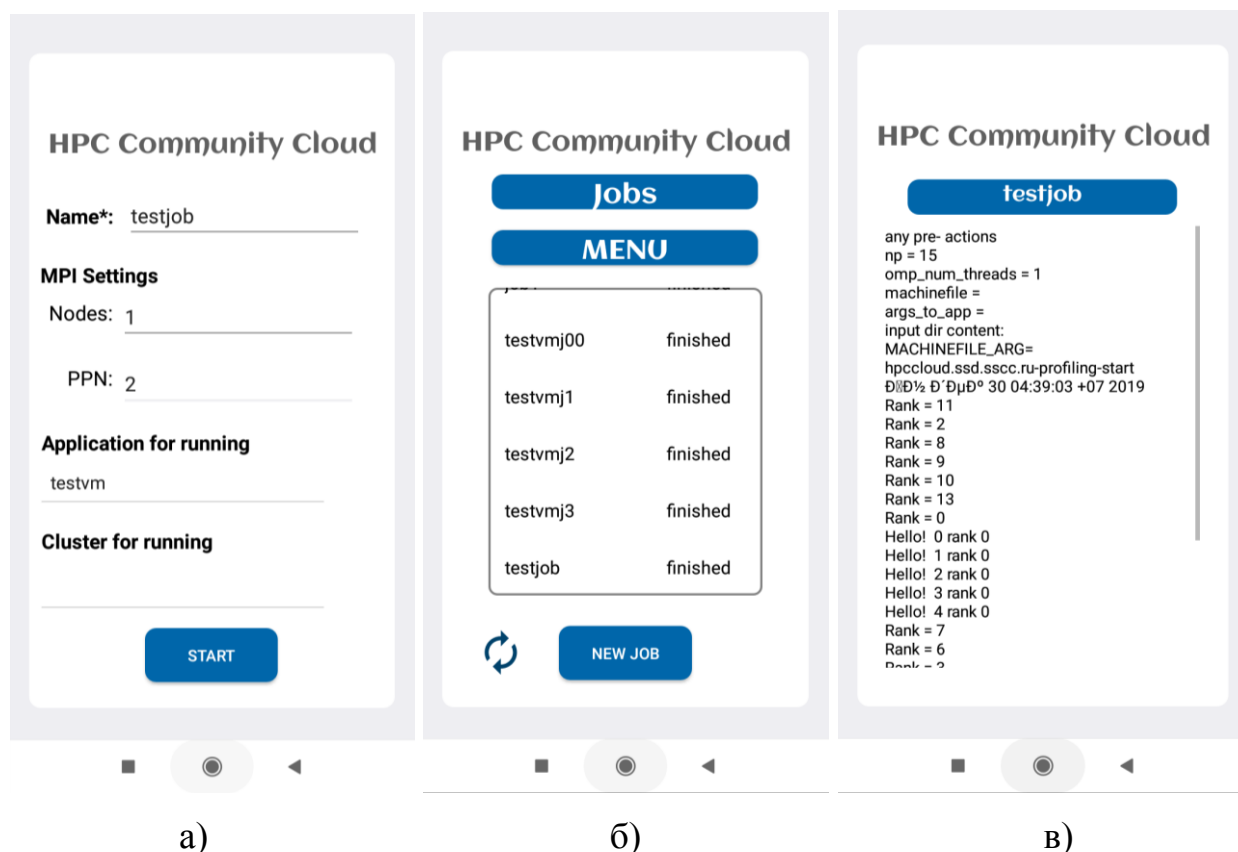


Рисунок 39 – Постановка новой задачи на выполнение (а), её отображение в общем списке задач (б) и просмотр результатов вычисления (в)

3.4.2. Профилировщик под управлением HPC2C

Доступ к модулю профилировщика предоставляется пользователю после авторизации в приложении через пункт меню, отображающий список доступных пользователю приложений (рисунок 38а)), где нажав кнопку Measure application performance пользователь переходит в окно работы с профилировщиком (рисунок 40а)). В данном окне необходимо ввести данные для оценки производительности и запустить профилировщик (рисунок 40б)). После выполнения требуемых действий пользователю будет предоставлена информация о результатах оценки приложения (рисунок 40в)).

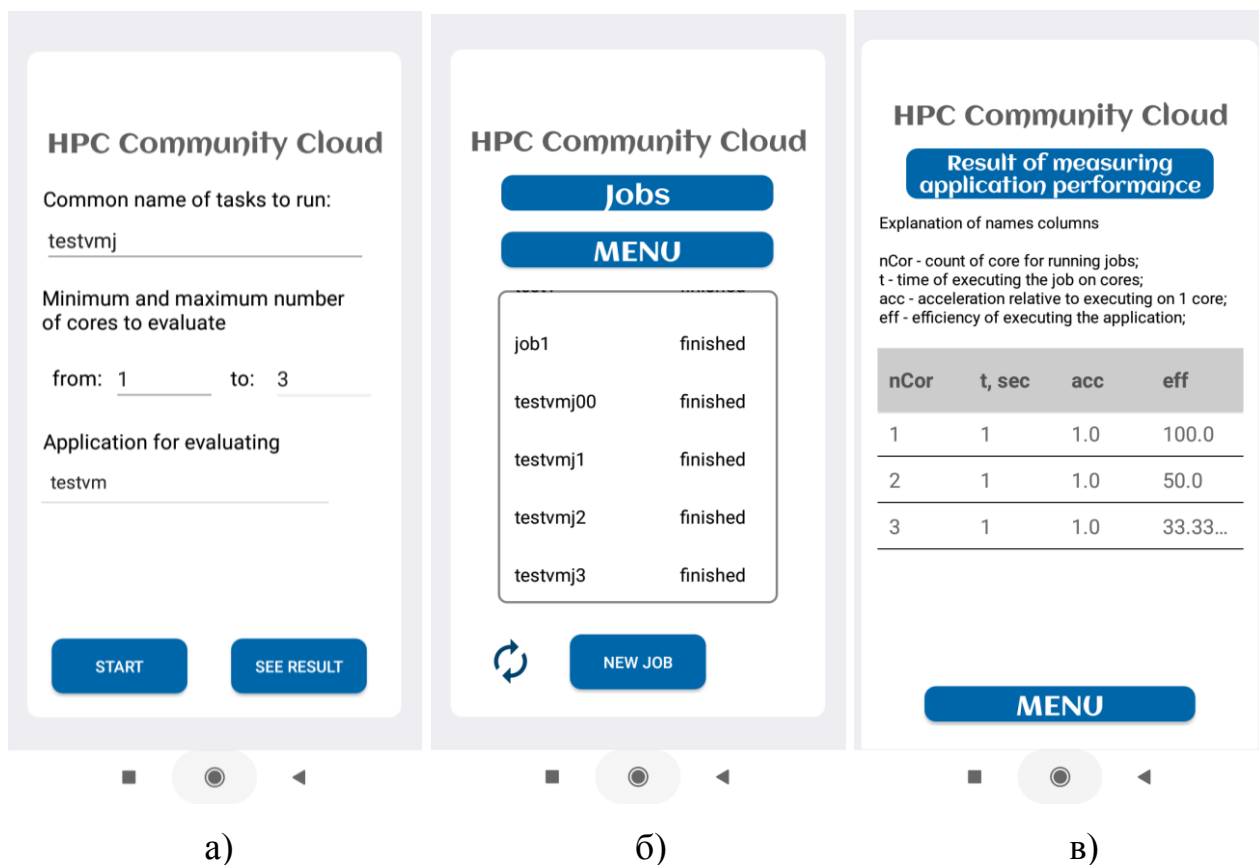


Рисунок 40 – Установка параметров профилировщика (а), отображение оцениваемых задач в общем списке (б) и результат работы профайлера (в)

ЗАКЛЮЧЕНИЕ

В результате работы были получены следующие результаты:

- спроектирована программная система для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах под управлением HPC Community Cloud (HPC2C); компоненты программной системы: мобильный клиент сервиса HPC2C, модуль оценки производительности вычислительного приложения, классификация пользовательских объектов, конструктор типов на основе классификации, генератор форм для входных переменных вычислительных приложений на основе присвоенных им типов;
- реализованы компоненты программной системы:
 - спроектирована классификация типов, разделяющая используемые внутри программной системы типы на несколько классов, имеющих характерные отличительные признаки;
 - спроектирован и реализован конструктор типов, позволяющий на основе классификации пользовательских объектов, используемых при создании вычислительных сценариев, описывать новые типы в виде JSON-схем;
 - реализован генератор форм, позволяющий на основе JSON-схемы, соответствующей каждому типу, генерировать формы для объектов с такими типами;
 - спроектирован и реализован прототип мобильного приложения, предоставляющего доступ к базовым операциям работы с сервисом HPC Community Cloud и возможностью дальнейшего расширения функционала;
 - реализован дополнительный модуль оценки производительности вычислительного приложения с помощью сервиса HPC2C, встроенный в мобильное приложение;

- на примере клеточно-автоматной модели показана методика разработки пользовательского интерфейса с использованием предлагаемых средств, в том числе:
 - разработана вычислительная модель FHP-MP, моделирующая течение газа в трубе;
 - специфицированы типы, используемые в разработанной модели с помощью конструктора типов;
 - реализованы операции визуализации данных соответствующих типов на основе JSON-схем;
 - на основе спецификаций задачи получения начального состояния клеточного автомата сгенерированы интерфейсы входных объектов.

В дальнейшем планируется расширение функциональности систем, путём добавления недостающих интерфейсов, включаемых в базовую функциональность системы: добавление десктопного интерфейса для взаимодействия с сервисом HPC2C, а также создание мобильного интерфейса для операционной системы IOS. Кроме того планируется наращивание библиотеки готовых плагинов для визуализации результатов вычислений и добавление возможности совместного управления вычислениями, а также предоставление личного или группового доступа к проектам пользователя.

СПИСОК ЛИТЕРАТУРЫ

1. MPI: A Message-Passing Interface Standard. Version 3.1. University of Tennessee, Knoxville, Tennessee, June 4, 2015. — 868 p.
2. Информационно-вычислительный центр Новосибирского государственного университета. [Электронный ресурс]. URL: <http://nusc.nsu.ru/wiki/doku.php/doc/index> (дата обращения: 15.06.2020).
3. Altair PBS Pro: Altair PBS Professional User's Guide. Version 13.0. Altair Engineering, Inc. 2003-2015. — 339 p.
4. Altair PBS Pro: Altair PBS Professional Reference Guide. Version 13.0. Altair Engineering, Inc. 2003-2015. — 534 p.
5. Оленёв Н.Н., Печёнкин Р.В., Чернецов А.М. Параллельное программирование в MATLAB и его приложения: монография / Н.Н. Оленёв, Р.В. Печёнкин, А.М. Чернецов — Москва: ВЦ РАН, 2007. — 120 с.
6. Пономарева, И.С. Некоторые аспекты создания web-приложения на базе MATLAB Web Server / И.С. Пономарева, В.А. Зелепухина, Ю.Ю. Тарасевич // Информационные технологии, № 9, 2006 — с. 68-72.
7. MATLAB Distributed Computing Server: Perform MATLAB and Simulink computations on clusters, clouds, and grids. [Электронный ресурс]. URL: <https://uk.mathworks.com/products/distriben.html> (дата обращения: 15.06.2020).
8. Межведомственный Суперкомпьютерный Центр Российской Академии Наук (МСЦ РАН). [Электронный ресурс]. URL: <http://www.jscc.ru/scomputers.html> (дата обращения: 15.06.2020).
9. Технологии параллельного программирования: НОРМА | PARALLEL.RU – Информационно-аналитический центр по параллельным вычислениям. [Электронный ресурс]. URL: <https://parallel.ru/tech/norma/> (дата обращения: 15.06.2020).
10. Andrianov A. NONPROCEDURAL NORMA LANGUAGE AND ITS TRANSLATION METHODS FOR PARALLEL ARCHITECTURES / A.N.

- Andrianov, T.P. Baranova, A.B. Bugerya, K.N. Efimkin // University News. North-Caucasian Region. Technical Sciences Series. 2017. № 3. p. 5–12.
11. Малышкин В.Э. Параллельное программирование мультимикомпьютеров. / В.Э.Малышкин, В.Д.Корнеев. – В серии «Учебники НГТУ», Новосибирск, изд-во НГТУ, 2006 – 296 с.
12. Нариньяни А.С. Модель или алгоритм – новая парадигма информационной технологии. Информационные технологии, №4, 1997. с.18-22.
13. OWL 2 Web Ontology Language Document Overview (Second Edition). [Электронный ресурс]. URL: <https://www.w3.org/TR/2012/REC-owl2-overview-20121211/> (дата обращения: 15.06.2020).
14. Городничев М.А., Вайцель С.А. Организация доступа к высокопроизводительным вычислительным ресурсам в HPC Community Cloud. Вестник ЮУрГУ. Серия: Вычислительная математика и информатика, 2014, Том. 3, №. 4, p. 85–95.
15. Городничев М.А., Комиссаров А.В., Можина А.В., Прочкин П.В., Рудыч П.Д., Юрченко А.В. Модели и проектные решения системы хранения и обработки исследовательских данных Ecclesia. Вестник НГУ. Серия: Информационные Технологии, 2018, Том. 16, №. 3, p. 87–104.
16. TensorFlow. [Электронный ресурс]. URL: <https://www.tensorflow.org/> (дата обращения: 15.06.2020).
17. Orange Data Mining – Data Mining. [Электронный ресурс]. URL: <https://orange.biolab.si/> (дата обращения: 15.06.2020).
18. Аналитическая платформа Loginom. [Электронный ресурс]. URL: <https://loginom.ru/> (дата обращения: 15.06.2020).
19. ProActive Workflows & Scheduling - Job Scheduler. [Электронный ресурс]. URL: <https://www.activeeon.com/products/workflows-scheduling/> (дата обращения: 15.06.2020).
20. Luna. A WYSIWYG language for data processing. [Электронный ресурс]. URL: <https://www.luna-lang.org/> (дата обращения: 15.06.2020).

- 21.Мошкина А.Д. Разработка программной системы для реализации пользовательских интерфейсов вычислительных приложений, работающих на высокопроизводительных вычислительных системах : выпускная квалификационная работа, научный руководитель Маркова Валентина Петровна – Новосибирск. – 2018. – С. 96.
- 22.Вальковский В. А., Малышкин В. Э. Синтез параллельных программ и систем на вычислительных моделях. – Новосибирск: Наука, 1988. – 129 с.
- 23.Лайкова, А. А. Юзабилити сайта: принципы и методы оценки / А. А. Лайкова // Актуальные проблемы авиации и космонавтики. - 2016. - № 12. -С. 745-747.
- 24.Gorodnichev M.A., Medvedev Y. A web-based platform for interactive parameter study of large-scale lattice Gas automata // Lecture Notes in Computer Science. – 2019. – Vol. 11657 : PaCT 2019: Parallel Computing Technologies . – P. 321-333. – DOI: 10.1007/978-3-030-25636-4_25.
- 25.Medvedev Yu.G. Lattice gas Cellular Automata for a ow simulation and their parallel implementation // Parallel Programming: Practical Aspects, Models and Current Limitations. Series: Mathematics Research Developments. Tarkov MS (ed) Hauppauge, New York: NSP Inc. – 2014. Pp. 143-158.
- 26.Medvedev Yu.G. Cellular-Automaton Simulation of a Cumulative Jet Formation / Malyshkin V. (eds) // PaCT-2009. LNCS, vol 5698. Springer, Berlin, Heidelberg. – 2009. Pp. 249-256. https://doi.org/10.1007/978-3-642-03275-2_25
- 27.Antonov A.S., Volkov N.I. An AlgoView Web-visualization System for the AlgoWiki Project / A.S. Antonov, N.I. Volkov // Parallel Computational Technologies (PCT'2017) – Kazan, Russia, 2017. – p. 3-13.
- 28.Грибова В.В. Автоматизация разработки пользовательских интерфейсов с динамическими данными / В.В. Грибова, Н.Н. Черкезишвили // Откры-

- тые семантические технологии проектирования интеллектуальных систем OSTIS-2011, Материалы I Международной научно-технической конференции 10 февраля 2011 г. Минск. – Минск: БГУИР, 2011. – с. 287-292.
- 29.Генаев М. А., Комышев Е. Г., Гунбин К. В., Афонников Д. А. BioInfoWF – система автоматической генерации Web-интерфейсов и Web-сервисов для биоинформационных исследований / Вавиловский журнал генетики и селекции. 2012. Т. 16. №4/1. – с. 849.
- 30.Зелепухина В.А. Разработка систем управления содержимым интернет-ресурсов на основе автоматической генерации WEB-интерфейса и SQL-запросов / В.А. Зелепухина // Информационные технологии, № 8, 2008 – с. 20-22.
- 31.Грибова В.В., Клещев А.С. Методы и средства разработки пользовательского интерфейса: современное состояние // Программные продукты и системы, 2001. №1. – с. 2-6.
- 32.Грибова В.В., Клещев А.С. Управление проектированием и реализацией пользовательского интерфейса на основе онтологий // Проблемы управления №2. 2006. – с.58-62.

Приложение А

Запуск вычислительной задачи на кластере НГУ

(примечание – цветом выделены комментарии)

```
# Указывается путь к bash-интерпретатору
#!/bin/bash

# Строки, начинающиеся с "#PBS", содержат директивы, используемые планировщиком. Ресурсы, относящиеся к серверу PBS или к очереди, используются всей задачей и в запросе указываются в следующем виде: "-l Ресурс=Значение"
# Время, необходимое задаче для работы
#PBS -l walltime=00:01:00

# Ресурсы, относящиеся к узлам (или виртуальным узлам), запрашиваются и выделяются в виде блоков (chunk). Запрос в большинстве случаев можно представить в следующем виде: "-l select=N:chunk", где строка "chunk" имеет вид "Ресурс1=Значение1[:Ресурс2=Значение2 ...]", а число "N" - сколько таких блоков требуется.
# Задаче необходимы два блока, каждый из которых содержит 4 ядра и 2000 МБ памяти, и на каждом из этих блоков будут запущены 4 MPI процесса, причём расположение выделяемых блоков будет произвольно (place=free)
#PBS -l select=2:ncpus=4:mpiprocs=4:mem=2000m,place=free

# Переход в каталог, на который указывает переменная окружения $PBS_O_WORKDIR
cd $PBS_O_WORKDIR

# Подсчёт суммарного количества MPI процессов, требующих запуска
MPI_NP=$(wc -l $PBS_NODEFILE | awk '{ print $1 }')

# Вывод на экран числа MPI процессов, хранящееся в переменной $MPI_NP
echo "Number of MPI process: $MPI_NP"

# Вывод на экран сообщения "File $PBS_NODEFILE:"
echo 'File $PBS_NODEFILE:'

# Вывод на экран содержимого файла, на который указывает переменная окружения $PBS_NODEFILE
cat $PBS_NODEFILE

# Вывод на экран пустой строки
echo

# Запуск программы с помощью команды trigrun, которая узнаёт выделенные задаче узлы из файла, на который указывает переменная окружения $PBS_NODEFILE (для intel_mpi ключ "-hostfile" необходимо заменить на "-machinefile")
mpirun -hostfile $PBS_NODEFILE -np $MPI_NP ./task.comp
```

Приложение Б

Запуск задач на суперкомпьютере МВС-10П МСЦ РАН с использованием планировщика SLURM в пакетном режиме с помощью утилиты srun

(примечание – цветом выделены комментарии)

Указывается путь к sh-интерпретатору

#!/bin/sh

Указание ограничения по времени, равное 1 минуте

#SBATCH --time=1

*# Запуск задачи с помощью утилиты srun, где на каждом вычислительном узле необходимо выделить X узлов, при общем количестве узлов X*Y*

srun --ntasks-per-node X -n X*Y ./task.comp

Приложение В

Запуск задач на суперкомпьютере МВС-10П МСЦ РАН с использованием планировщика SLURM в пакетном режиме с помощью утилиты `mpirun.hydra`

(примечание – цветом выделены комментарии)

Указывается путь к sh-интерпретатору

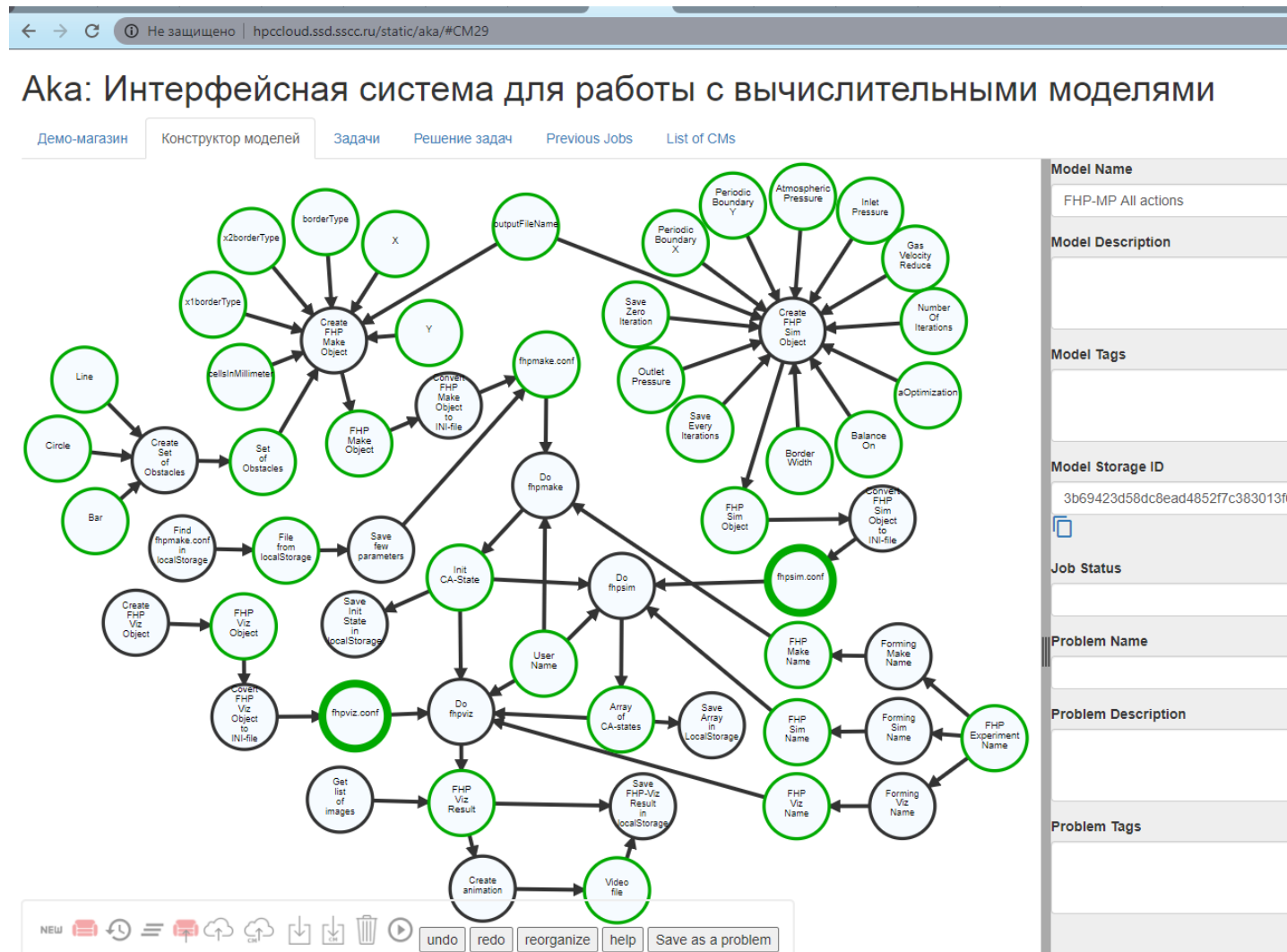
`#!/bin/sh`

*# Запуск задачи с помощью утилиты `mpirun.hydra`, где на каждом вычислительном узле необходимо выделить X узлов, при общем количестве узлов $X*Y$*

`mpirun.hydra -perhost X -n X*Y ./task.comp`

Приложение Г

Вид системы клеточно-автоматного моделирования в виде вычислительного сценария



Приложение Д

Пример файла конфигурации fhpmake.conf

```
# FHP-MP constructor configuration

[main]
outputFileName=a.ca
X=2000
Y=1000
cellsInMillimeter=1
borderType=wall
x1borderType=inlet
x2borderType=outlet

[circle1]
x=1000
y=500
r=1120
cellsType=conventional
gasConcentration=15
gasRestConcentration=0

[bar1]
x1=0
y1=0
x2=2000
y2=1000
cellsType=conventional
gasConcentration=5
gasRestConcentration=0

[line5]
x1=0
y1=0
x2=2000
y2=0
cellsType=wall

[line6]
x1=0
y1=1000
x2=2000
y2=1000
cellsType=wall

[line3]
x1=0
y1=0
x2=0
y2=1000
cellsType=inlet

[line4]
x1=2000
y1=0
x2=2000
y2=1000
cellsType=outlet

[line11]
x1=450
y1=650
x2=550
y2=350
cellsType=wall

[circle10]
x=400
y=500
r=50
cellsType=wall
gasConcentration=0
gasRestConcentration=0
```

Приложение Е

Отображение окон интерфейса для вариантов переменной Set of Obstacles

Интерфейс сгенерированный для препятствия в виде круга

Set of Obstacles

Set of Obstacles

Add Item

Delete Item

Item 0 of Set of Obstacles

Select type of item:

Circle

▼

Circle

x

y

r

cellsType

conventional

▼

gasConcentration

gasRestConcentration

Интерфейс сгенерированный для препятствия в виде прямоугольника или квадрата

Set of Obstacles

Add Item

Delete Item

Item 0 of Set of Obstacles

Select type of item:

Bar

Bar

x1

y1

x2

y2

cellsType

conventional

gasConcentration

gasRestConcentration

Интерфейс сгенерированный для препятствия в виде линии

Set of Obstacles

Add Item

Delete Item

Item 0 of Set of Obstacles

Select type of item:

Line

Line

x1

y1

x2

y2

cellsType

conventional

gasConcentration

gasRestConcentration