Метод преобразования архитектуры программы построения графиков из централизованной в клиент-серверную

П. В. Егоров[1]

[1]ФГУ ФНЦ НИИСИ РАН, Москва, Россия, egorov@niisi.ras.ru, +7(903) 552-48-87

Аннотация. Применительно к программе построения графиков разработан метод преобразования однопользовательского приложения с централизованной архитектурой в многопользовательское приложение с архитектурой «клиент-сервер». Программа построения графиков входит в состав программной системы экологического мониторинга окружающей среды, разработанной в ФГУ ФНЦ НИИСИ РАН. В системе применяется встроенная СУБД, не поддерживающая технологию «клиент-сервер». Проведен реинжиниринг программы. В переработанной программе архитектура «клиент-сервер» реализована с использованием веб-технологий. В веб-приложении используется свободная программа Spring Framework.

Ключевые слова: централизованная архитектура, архитектура «клиент-сервер», экологический мониторинг окружающей среды, Spring Framework, веб-сервер, веб-приложение

1. Введение

В статье рассматривается метод преобразования однопользовательского приложения с централизованной архитектурой в многопользовательское приложение с архитектурой «клиент-сервер» применительно к программе построения графиков. Программа построения графиков входит в состав программной системы экологического мониторинга окружающей среды, разработанной в ФГУ ФНЦ НИИСИ РАН (далее – «система»). Система обеспечивает сбор данных с датчиков оборудования полигона твердых бытовых отходов, сохранение их в базе данных (далее БД), построение отчетов и графиков на основе данных из БД. Графики отображают динамику изменения параметров выбросов загрязняющих веществ на объекте автоматизации.

Далее под централизованной архитектурой будет пониматься такая архитектура программной системы, когда все ее компоненты располагаются на одной ЭВМ.

Определение архитектуры «клиент-сервер» приведено в [1]. Такая архитектура обеспечивает удаленный доступ к ресурсам сервера, а также многопользовательский режим работы.

Система состоит из следующих программных компонент:

- сервер ввода-вывода, который обеспечивает прием и сохранение в долговременной памяти данных с датчиков,
- сервер данных, который выполняет усреднение данных с датчиков на временном интервале в одну минуту и запись их в базу данных,
- программа формирования отчетов,
- программа построения графиков (ППГ).

При проектировании программы построения графиков предполагалось, что приложение будет устанавливаться на персональной ЭВМ, расположенной на объекте автоматизации, и обслуживать одного оператора. В процессе опытной эксплуатации системы возникла необходимость обеспечить просмотр графиков с персональных компьютеров, подключенных к ЭВМ на объекте автоматизации (далее сервер) по сети. В качестве способа решения этой задачи было выбрано преобразование архитектуры программы в клиент-серверную с помощью веб-технологий.

Одним из требований к новой архитектуре программы была поддержка СУБД «SQLite» для сохранения совместимости с программой сервера данных. Особенностью СУБД «SQLite» является то, что она не поддерживает технологию «клиент-сервер» [2] (далее – «встроенная СУБД»)

В статье будет описан метод создания программы построения графиков с архитектурой «клиент-сервер», в которой используется встроенная СУБД «SQLite». Одним из преимуществ данного метода является то, что он позволяет повторно использовать программный код исходной программы построения графиков и сохранить программную совместимость ППГ с компонентами системы мониторинга экологических параметров окружающей среды.

Подход, основанный на веб-технологиях, обладает следующими преимуществами.

1. Распределенность. Пользователь может работать с веб-приложением с любой ПЭВМ, на которой запущен веб-браузер, и которая имеет сетевой доступ к веб–серверу. Далее веб-приложением будем называть программную систему, в которой пользователь взаимодействует с веб-сервером при помощи браузера [3].

2. Переносимость. Браузеры, такие как «Яндекс Браузер», «Google Chrome», свободно распространяются и могут быть установлены как на ПЭВМ под управлением операционных систем (ОС) семейства Windows и Linux, так и на цифровые мобильные устройства под управлением ОС Android. Дистрибутивы большинства серверных ОС включают в себя веб–сервер. Некоторые веб-серверы, такие как Apache HTTP-сервер, являются свободным программным обеспечением. Для создания веб-приложений разработаны переносимые свободно распространяемые языки программирования, такие как Java или PHP.

3. Стандартный графический интерфейс пользователя, который реализует браузер.

4. Простота установки и обслуживания. Программа построения графиков устанавливается на сервер. На ПЭВМ и мобильных цифровых устройствах браузер обычно предустановлен.

В данной работе моделирование программной архитектуры осуществляется с помощью методов формализации, абстрагирования, алгоритмизации и тестирования, прямого и обратного проектирования [4].

В качестве средства формализации используется язык моделирования UML [5].

Основные термины, используемые при описании моделей, и соглашения по обозначениям элементов модели в тексте были рассмотрены в работе [6]

2. Описание централизованной архитектуры программы построения графиков

Диаграмма классов программы с централизованной архитектурой показана на рисунке 1.

Программная архитектура по функциональному признаку делится на три уровня:

- Графический интерфейс – реализует построение графического пользовательского интерфейса. На этом уровне находится класс «ГПИ» и пакет «Swing».

- Построение графика – реализует логику построения графиков. Этот уровень содержит класс «График» и пакет «JFreeChart».

- Управление данными –этот уровень отвечает за работу с БД. На этом уровне находится класс «БД».

Класс «График» описывает объекты, реализующие функцию построения графика. Класс имеет два метода build() и getChart(). Операция build() предназначена для построения графика заданного типа в целевом формате данных. Метод getChart() возвращает данные графика в целевом формате, а именно в программе построения графиков она возвращает объект типа ChartPanel, с которым связаны операции рисования классов библиотеки JFreeChart. Класс «ChartPanel» является наследником класса «JPanel» библиотеки «Swing» и, как и он, является контейнером графических объектов. Графическое пространство на экране дисплея, принадлежащее объекту «:*ChartPanel*», далее будем называть панелью. На рисунке 3 панель объекта «:*ChartPanel*» находится наверху окна программы и содержит изображение графика.

Реализацию операции build() класс «График» делегирует классам библиотеки JFreeChart, что на диаграмме показано с помощью отношения зависимости «build». Библиотека JFreeChart обозначена пакетом «JFreeChart».

JFreeChart — это бесплатная библиотека для языка Java, которая включает API, поддерживающий широкий спектр типов диаграмм. Она поддерживает типы выходных данных, такие как компоненты Swing и JavaFX, файлы изображений (включая PNG и JPEG) и форматы файлов векторной графики (включая PDF, EPS и SVG) [7].

Класс «БД» описывает объект, обеспечивающий выборку данных из БД, в которой хранится информация с датчиков. Этот класс реализует операцию getData(), которая возвращает набор данных в соответствии с заданными условиями поиска.

Метод getData() объекта «:*БД*» вызывается объектом «:*График*» для загрузки данных, необходимых ему для построения графика (см. рисунок 2). На диаграмме вызов метода обозначен отношением зависимости «getData».

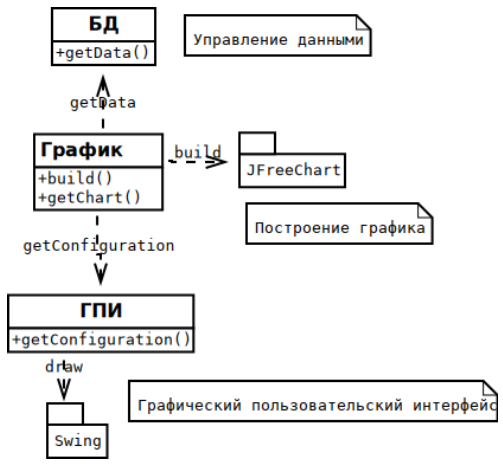

Рис. 1. Централизованная архитектура программы построения графиков

Для каждого графика имеется список конфигурационных параметров, которые определяют набор данных для построения графика, например, тип графика, период времени за который строится график (см. рисунок 3).

Конфигурационные параметры вводятся в программу оператором с помощью графического пользовательского интерфейса (далее ГПИ).

На диаграмме графический пользовательский интерфейс обозначен классом «ГПИ». У класса имеется операция getConfiguration(), которая позволяет получить набор конфигурационных данных, введенных в программу оператором. Этот метод вызывается объектом *«:График»* для получения конфигурационных данных, что на диаграмме показано с помощью отношения зависимости «getConfiguration».

Объект «*:ГПИ*» делегирует функции построения графического интерфейса объектам классов библиотеки Swing, что на диаграмме обозначено с помощью отношения зависимости «draw». Библиотека Swing обозначена пакетом «Swing».

Swing — это библиотека для создания графического интерфейса на языке Java. Она содержит ряд графических компонентов, таких как кнопки, поля ввода, таблицы и т. д. [8].

3. Функционирование программы построения графиков с централизованной архитектурой

На рисунке 2 показана диаграмма взаимодействия объектов программы.

Поток управления программы построения графика можно представить следующей последовательностью команд.

«:*Оператор*» с помощью объекта «:*ГПИ*» вводит конфигурационные данные графика в программу (см. рисунок 3) и инициирует процесс построения графика передачей сообщения start().

При получении сообщения start() объект «:*ГПИ*» командой new() создает объект «:*График*» и затем вызывает метод getChart(), который возвращает объект «:*ChartPanel*». Объект «:*График*» с помощью сообщения getConfiguration() получает от объекта «:*ГПИ*» конфигурационные данные графика. Затем объект «:*График*» вызывает метод getData() объекта «:*БД*» для загрузки данных датчиков за определенный период времени, определяемый конфигурационными данными. После формирования указанных выше наборов данных объект «:*График*» вызывает метод build(), который формирует данные цифрового изображения графика и инициирует отрисовку изображения объектами классов пакета «JFreeChart» на панели объекта «:*ChartPanel*».

Пример фрагмента окна программы с графиком концентрации сероводорода показан на рисунке 3.

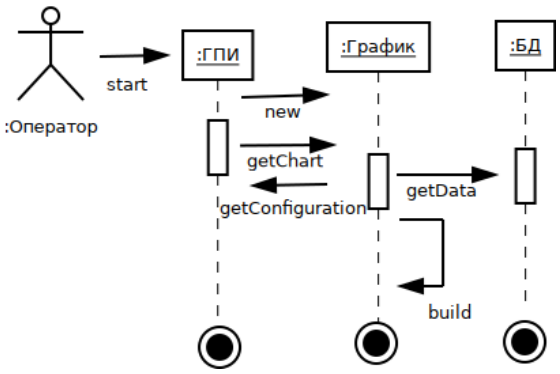

Рис. 2. Диаграмма взаимодействия

4. Диаграмма развертывания программы с централизованной архитектурой

Программа построения графиков устанавливается на персональный компьютер под управлением ОС Fedora 35, расположенный на объекте автоматизации. На рисунке 4 показана диаграмма развертывания программы на ПК.

На диаграмме показаны следующие программные компоненты: «ППГ», «JDK» и «SQLite».

Компонент «SQLite» обозначает СУБД SQLite, которая управляет БД усредненных показаний датчиков.

Компонент “ППГ” обозначает программу построения графиков.

С помощью отношения агрегирования между компонентами «ППГ» и «SQLite» показано, что СУБД SQLite является встроенной, то есть компилируется вместе с программой построения графиков в один исполняемый файл.

Код программы построения графиков реализован на языке Java, и для его выполнения требуется, чтобы на ПЭВМ был установлен комплект разработчика приложений (JDK) [9], который на диаграмме показан с помощью компонента «JDK».

Персональный компьютер рабочего места оператора обозначен на диаграмме в виде узла с названием «ПК».

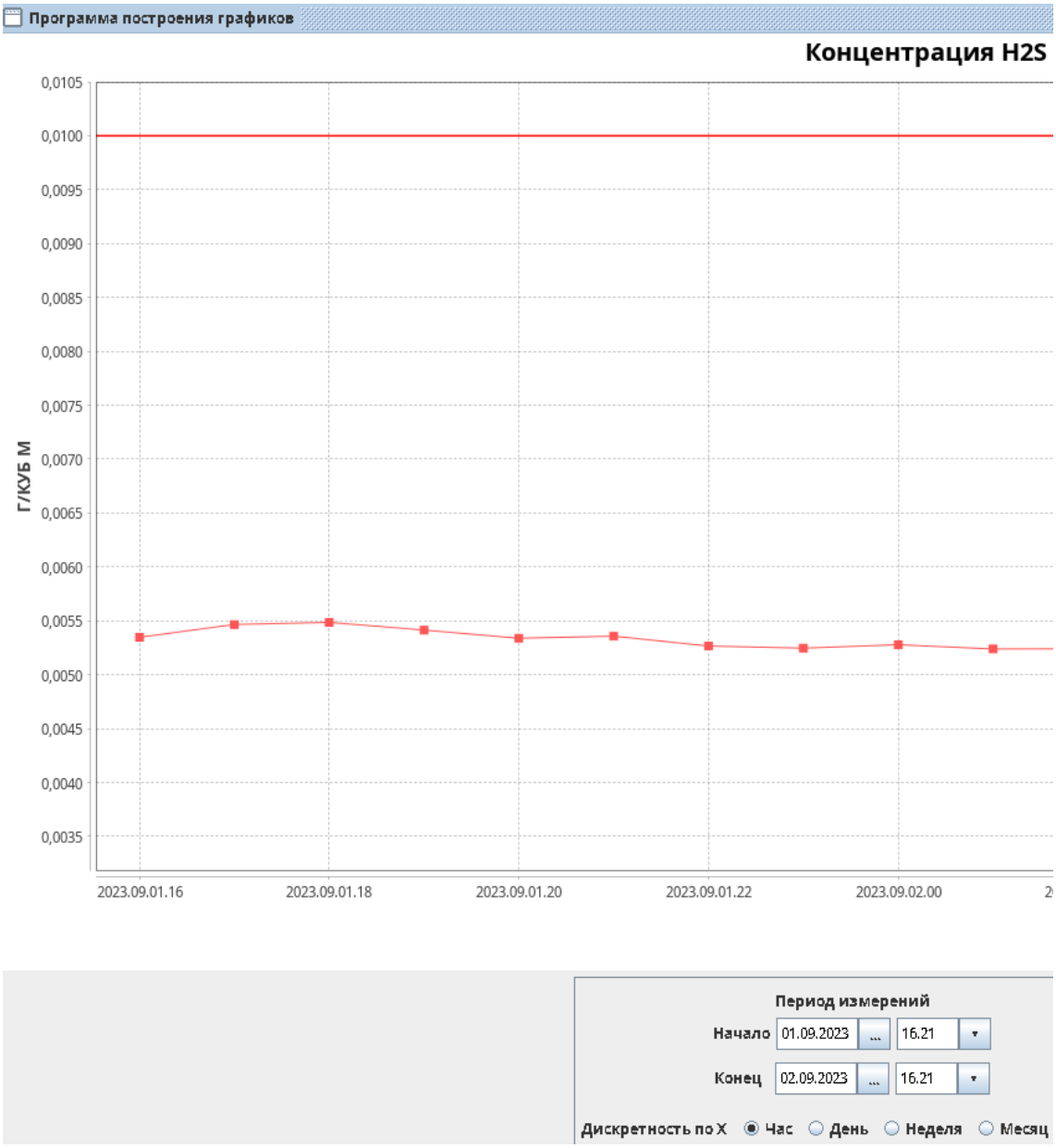

Рис. 3. Пример фрагмента окна программы с графиком концентрации сероводорода

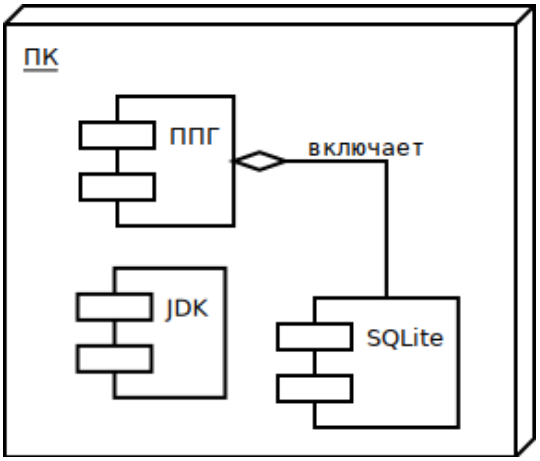

Рис. 4. Диаграмма развертывания программы

5. Описание метода преобразования централизованной архитектуры программы построения графиков в клиент-серверную

Программа построения графиков, реализованная в архитектуре «клиент-сервер», является распределенной. Ее могут эксплуатировать несколько пользователей на персональных компьютерах, объединенных в сеть.

Поскольку программа использует встроенную СУБД, которая не может выступать в роли сервера БД, для реализации архитектуры «клиент-сервер» были использованы веб-технологии. Преимущества такого выбора перечислены в разделе 1. На рисунке 5 показана диаграмма развертывания программы, которую далее будем называть веб-приложением.

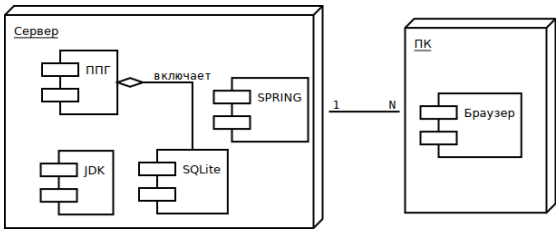

Рис. 5. Диаграмма развертывания веб-приложения

На аппаратном уровне архитектура системы включает в себя сервер и персональные компьютеры. На диаграмме соответствующие ЭВМ обозначены с помощью узлов «Сервер» и «ПК», ассоциация один ко многим показывает, что сервер может быть объединён сетью с одним или несколькими ПК.

На персональных компьютерах устанавливается браузер, который на диаграмме обозначен компонентом «Браузер». Браузер – это прикладное программное обеспечение для просмотра веб-страниц в распределенной системе WWW [10]. В веб-приложении браузер используется для построения графического интерфейса пользователя.

На сервере устанавливаются следующие программы: программа построения графиков (ППГ), JDK, СУБД SQLite и фреймворк с открытым исходным кодом для Java-платформы Spring Framework. Фреймворк – это программное обеспечение, облегчающее разработку и объединение разных компонентов большого программного проекта [11]. Все перечисленные компоненты, кроме Spring Framework были описаны в предыдущем разделе.

Программа Spring Framework создает компоненты приложения и управляет ими [12]. В состав таких компонент входят веб-сервер и программные средства аутентификации пользователя. Основными критериями при выборе этой программы были свободное распространение, кроссплатформенность, поддержка языка Java и веб-технологий. Эта программа имеет модульную структуру. В веб-приложении использовались следующие модули.

1. Модуль Spring Data JPA, который обеспечивает поддержку репозитория в соответствии со спецификацией Jakarta Persistence API (JPA) [13]. Репозиторий — это место, где хранятся и поддерживаются какие-либо данные [14]. Спецификация JPA определяет следующее:

- способ определения метаданных отображений.
- API для основных CRUD-операций (создать, прочесть, обновить, удалить).
- язык и API для создания запросов, использующих классы и их свойства.
- порядок взаимодействия механизма хранения с транзакционными сущностями [15].

2. Модуль MVC [16] является веб-фреймворком, предназначенным для создания динамических веб-сайтов, сетевых приложений, сервисов или ресурсов [17]. Модуль MVC определяет схему разделения данных приложения и управляющей логики на три отдельных компонента: модель, представление и контроллер — таким образом, что модификация каждого компонента может осуществляться независимо.

Модель предоставляет данные и реагирует на команды контроллера, изменяя своё состояние.

Представление отвечает за отображение данных модели пользователю, реагируя на изменения модели.

Контроллер интерпретирует действия пользователя, оповещая модель о необходимости изменений [18].

3. Модуль Spring Security предоставляет механизмы построения систем аутентификации и авторизации для приложений, созданных с помощью Spring Framework. [19].

4. Модуль Spring Boot — это расширение, которое упрощает и ускоряет работу со Spring. Оно представляет собой набор утилит, автоматизирующих настройки фреймворка [20].

6. Архитектура веб-приложения

Архитектуру веб-приложения так же, как и централизованную архитектуру ППГ, можно логически разделить на три функциональных уровня: построение графика, управление данными и графический пользовательский интерфейс.

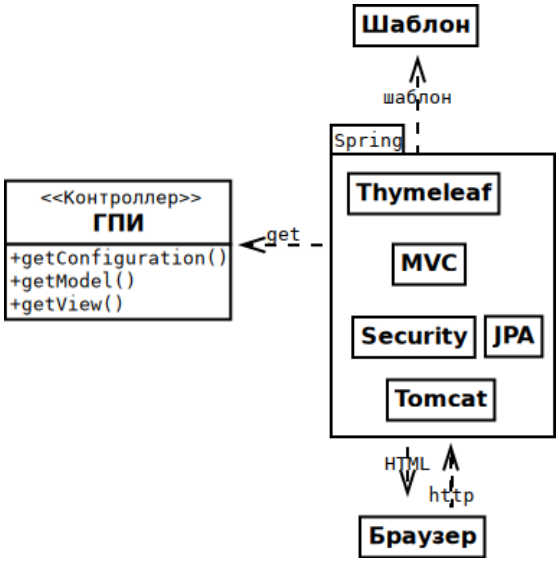

Рис. 6. Диаграмма классов уровня ГПИ веб-приложения

Модель уровней построение графика и управление данными в этих архитектурах одинаковая, поэтому в данном разделе она рассматриваться не будет.

На уровне графического пользовательского интерфейса модели архитектур веб-приложения и ППГ различаются. В модели веб-приложения (см. рисунок 6) отсутствует пакет «Swing» и добавлены класс «Браузер» и пакет «Spring».

Браузер на диаграмме обозначен классом «Браузер». Для построения интерфейса объекту «:*Браузер*» надо передать описание интерфейса на языке HTML, которое далее будем называть веб-страницей. На диаграмме загрузка веб-страницы объектом «:*Браузер*» показана с помощью отношения зависимости «HTML».

Кроме того, объект «:*Браузер*» осуществляет передачу http-запросов, сформированных с помощью ГПИ, объекту «:*Tomcat*», что на диаграмме показано отношением «http».

Пакет «Spring» обозначает программу Spring Framework со всеми модулями, перечисленными выше, а также дополнительными программами, необходимыми для функционирования веб-приложения.

Класс «Security» обозначает модуль Spring Security. В веб-приложении модуль обеспечивает функционал аутентификации пользователей, проверку безопасности входящих запросов.

Модуль «Spring Data JPA» на диаграмме обозначен классом «JPA». Он обеспечивает поддержку репозиториев пользователей и ролей. Данные из этих репозиториев используются модулем Spring Security в процессе аутентификации пользователей.

Класс «Tomcat» обозначает веб-сервер Tomcat. Настройку и запуск веб-сервера выполняет модуль Spring Boot (на диаграмме не показан).

Модуль MVC на диаграмме обозначен классом «MVC».

Класс «Thymeleaf» обозначает шаблонизатор Thymeleaf, который используется в веб-приложении для создания веб-страниц. Шаблонизатор — это программное обеспечение, позволяющее использовать шаблоны для генерации конечных документов с помощью декларативного языка разметки [21].

Класс «ГПИ» в структуре модуля MVC реализует функции контроллера, что на диаграмме показано с помощью стереотип класса «контроллер».

Класс «ГПИ» в веб-приложении помимо метода getConfiguration(), описанного в модели централизованной архитектуры, реализует метод getModel() и getView(). Метод getModel() возвращает модель данных, а метод getView() имя представления. Методы вызываются объектом «:*MVC*», что на диаграмме показано с помощью отношения зависимости «get».

Класс «Шаблон» обозначает входные данные шаблонизатора.

Классы «Шаблон» и «Thymeleaf» в структуре модуля MVC выполняют функции представления.

8. Описание функционирования веб-приложения

Основная задача веб-приложения – построение графика. Далее будет рассмотрена кооперация объектов веб-приложения, реализующая функцию построения графика.

Логически поток управления веб-приложения можно разделить на три части:

- авторизация пользователя,
- заполнение формы ввода параметров графика (далее форма) и преобразование их в конфигурационные данные,
- построение и отображение графика.

Авторизация пользователя начинается с заполнения формы с учетными данными. Для загрузки страницы с описанием формы авторизации оператор должен ввести в адресную строку браузера корректный адрес сайта и инициировать операцию отправки запроса.

В результате запрос от объекта «:*Браузер*» передается объекту «:*Tomcat*», который перенаправляет его объекту «:*Security*». Так как пользователь не авторизован и в его запросе отсутствует корректный идентификатор сессии, объект «:*Security*» отправит объекту «:*Браузер*» веб-страницу с описанием формы авторизации, которая в окне браузера имеет следующий вид (см. рисунок 7).

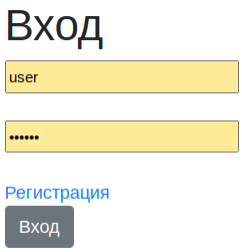

Рис. 7. Форма авторизации

«:*Оператор*» вводит учетные данные и инициирует передачу запроса от объекта «:*Браузер*» объекту «:*Tomcat*», а затем объекту «:*Security*». Для проверки данных запроса объект «:*Security*» передает поисковый запрос объекту «:*JPA*», чтобы найти имя пользователя, пароль, а также роли пользователя, которые задают его привилегии в веб-приложении. Если запись пользователя найдена, ее данные сравниваются с данными в запросе. Перед сравнением пароль из запроса кодируется целевым алгоритмом шифрования. Если операция сравнения прошла успешно, запрос проверяется на безопасность в соответствии с настройками модуля и передается объекту «:*MVC*». В настройках объекта «:*Security*» указан адрес страницы сайта, которую следует загрузить в браузер в случае успешной авторизации. Далее эту страницу будем называть главной. Объект «:*MVC*» определяет объект «:*ГПИ*», назначенный для построения главной страницы, и передает ему сообщения getModel() и getView(). Объект «:*ГПИ*» в ответ возвращает модель данных и имя шаблона. Объект «:*MVC*» передает эти данные объекту «:*Thymeleaf*», который по имени определяет объект «:*Шаблон*», получает от него данные и формирует веб-страницу. Затем объект «:*Thymeleaf*» с помощью объекта «:*MVC*» передает эту страницу объекту «:*Браузер*». В результате в окне браузера отобразится навигационное меню, фрагмент которого показан на рисунке 8. С помощью этого меню выполняется переход на страницу с формой для ввода параметров состояния целевого графика.

После успешной аутентификации пользователю выделяется идентификатор сеанса. В дальнейшем этот идентификатор сохраняется в файлах cookie и используется для идентификации пользователя при следующих запросах.[22].

Рис. 8. Фрагмент навигационного меню веб-приложения

Для заполнения формы необходимо загрузить в браузер соответствующую веб-страницу. Для загрузки веб-страницы с формой «:*Оператор*» активирует ссылку навигационного меню для одного из графиков в результате чего запрос передается от объекта «:*Браузер*» объекту «:*Tomcat*», а затем объекту «:*Security*».

Объект «:*Security*» проверяет идентификатор сеанса и, если проверка завершается успешно, передает запрос объекту «:*MVC*», который определяет объект «:*ГПИ*» для выполнения этого запроса. Дальнейшая последовательность операций по обработке запроса совпадает с той, что была описана выше для главной страницы.

В результате объект «:*Браузер*» загружает веб-страницу с описанием формы. Пример фрагмента страницы с формой ввода данных графика концентрации в окне браузера показан на рисунке 9.

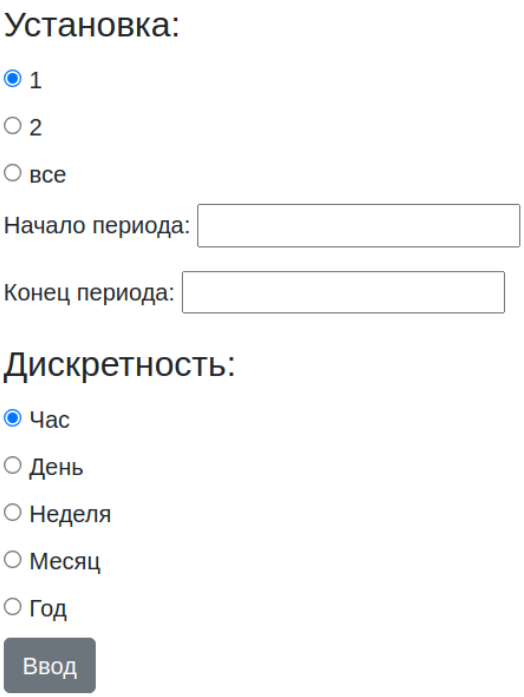

Рис. 9. Фрагмент формы ввода параметров графика концентрации

В веб-приложении применяется два способа формирования данных для загрузки в браузер.

Первый – это создание веб-страницы, как было описано выше. При втором способе формирования данных создается пакет данных с заголовком, определяющих их тип. Этот способ используется для загрузки в браузер изображения графика без перезагрузки веб-страницы и будет рассмотрен далее. При таком методе формирования данных не используются объекты «:*Thymeleaf*» и «:*Шаблон*».

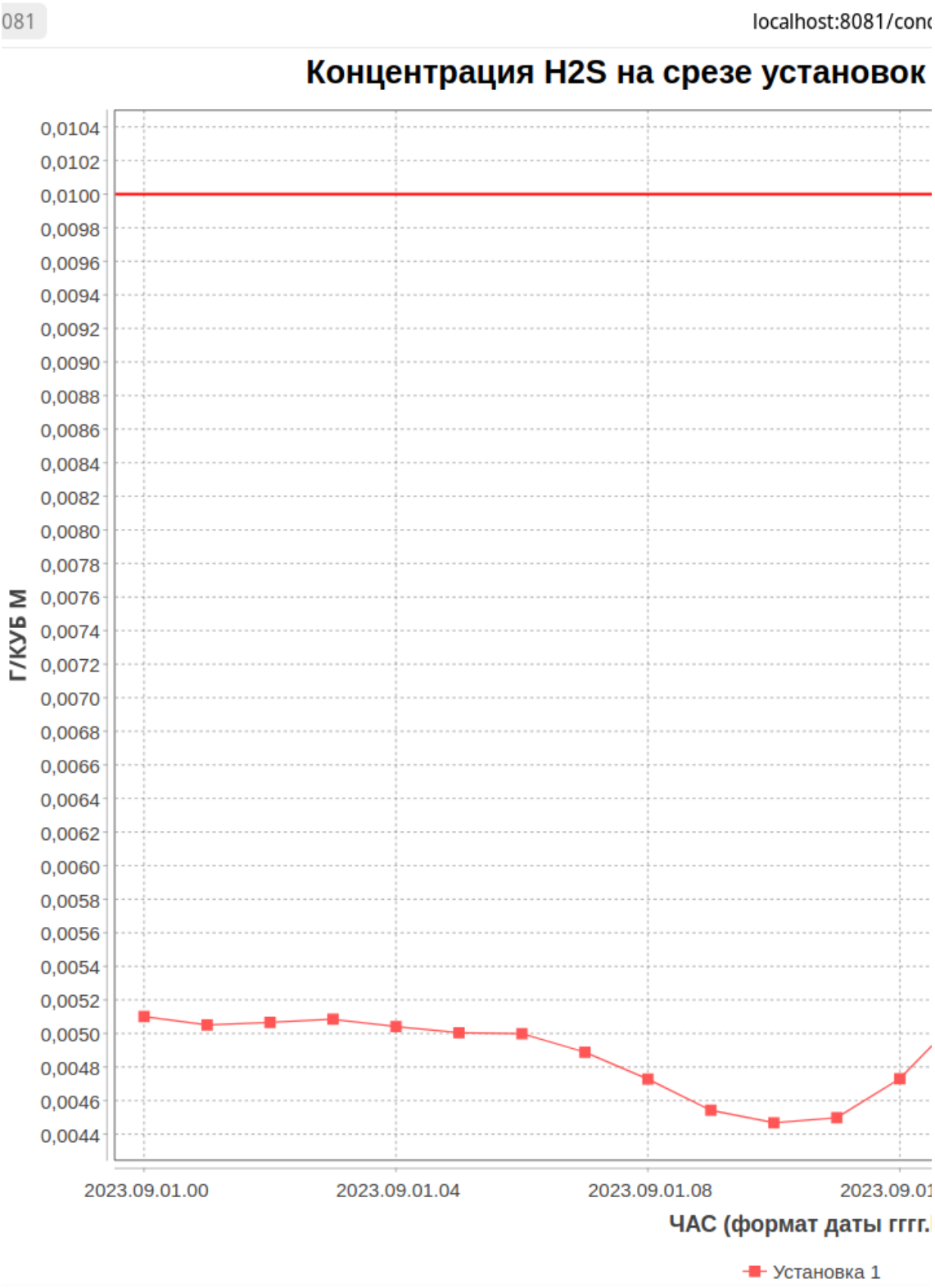

Рис. 10. Пример фрагмента окна браузера с изображением графика концентрации сероводорода

Построение графика начинается с ввода конфигурационных данных графика в форму. После заполнения формы «:*Оператор*» инициирует передачу запроса от объекта «:*Браузер*» объекту «:*Tomcat*», а затем объекту «:*Security*» для проверки идентификатора сеанса. Если проверка запроса завершилась успешно, запрос отправля-

ется объекту «:*MVC*», который определяет объект «:*ГПИ*» для обработки запроса и передает ему управление. Объект «:*ГПИ*» создает объект «:*График*» (см. рисунок 2), который с помощью методов getConfiguration() и getData() объектов «:*ГПИ*» и «:*БД*» формирует данные конфигурации графика и датчиков, затем вызывает метод build(), который на основе этих данных строит изображение графика. После этого объект «:*ГПИ*» получает данные графика в целевом формате с помощью метода getChart().

Полученные данные графика объект «:*ГПИ*» отправляет объекту «:*MVC*», который передает их объекту «:*Браузер*». На рисунке 10 показан фрагмент окна браузера с изображением графика концентрации сероводорода.

Заключение

В статье был рассмотрен метод преобразования централизованной архитектуры программы построения графиков в архитектуру «клиент-сервер» с применением веб-технологий.

Были рассмотрены модели программ с централизованной и клиент-серверной архитектурами с точки зрения типов данных, взаимодействия объектов и развертывания компонент программы на ПК.

Веб-технологии позволили реализовать архитектуру веб-приложения с применением СУБД SQLite, которая используется компонентами программной системы экологического мониторинга окружающей среды для управления данными, частью которой является программа построения графиков.

При реализации веб-приложения использовалась свободная программа Spring Framework, которая обеспечила реализацию функций авторизации пользователя и веб-сервера, тем самым уменьшив трудоемкость и время на создание веб-приложения.

Разработка веб-приложения предложенным в данной статье способом и эксплуатация программы у заказчика подтвердили правильность принятых решений с точки зрения архитектуры и реализации.

Публикация выполнена в рамках государственного задания по проведению фундаментальных исследований по теме «Исследование и реализация программной платформы для перспективных многоядерных процессоров» (FNEF-2022-002).»

Method for Converting the Architecture of a Chart Building Program from a Centralized to a Client-Server

P. V. Egorov

Abstract. A method of converting the chart building program architecture from a centralized to a client-server is considered. The program is a part of the software system for ecological monitoring of the environment, developed at the FSI FSC SRISA RAS.

Keywords: centralized architecture, client-server architecture, environmental monitoring, Spring Framework, web server

Литература

1. Клиент — сервер. https://ru.wikipedia.org/wiki/Клиент_—_сервер. (Дата обращения 08.09.2023).

2. SQLite Is Serverless. https://www.sqlite.org/serverless.html. (Дата обращения 28.08.2023).

3. Веб-приложение. https://ru.wikipedia.org/wiki/Веб-приложение. (Дата обращения 04.08.2023).

4. Reverse engineering. https://en.wikipedia.org/wiki/Reverse_engineering. (Дата обращения 08.09.2023).

5. Гради Буч, Джеймс Рамбо, Ивар Якобсон «Язык UML. Руководство пользователя». Москва «Издательство ДМК Пресс» 2006 г.

6. Егоров П. В. Описание метода построения библиотеки отображения растровой карты, инвариантной к форматам целевых геопространственных данных. Труды НИИСИ РАН. Том 11 N 4. Москва 2021 г.

7. Welcome To JFreeChart!. https://jfree.org/jfreechart (Дата обращения 04.08.2023).

8. Swing (библиотека). https://ru.wikipedia.org/wiki/Swing_(библиотека). (Дата обращения 04.08.2023).

9. Java Development Kit. https://ru.wikipedia.org/wiki/Java_Development_Kit. (Дата обращения 04.08.2023).

10. Браузер. https://ru.wikipedia.org/wiki/Браузер. (Дата обращения 08.09.2023).

11. Фреймворк. https://ru.wikipedia.org/wiki/Фреймворк. (Дата обращения 28.08.2023)

12. Уоллс К. Spring в действии. 6-е изд.– М.: ДМК Пресс, 2022.

13. Spring Data JPA - Reference Documentation. https://docs.spring.io/spring-data/jpa/docs/current/reference/html. (Дата обращения 04.08.2023).

14. Репозиторий. https://ru.wikipedia.org/wiki/Репозиторий. (Дата обращения 28.08.2023)

15. Бауэр К., Кинг Г., Грегори Г. Java Persistence API и Hibernate. – М.: ДМК Пресс, 2017.

16. Spring Web MVC. https://docs.spring.io/spring-framework/reference/web/webmvc.html. (Дата обращения 04.08.2023).

17. Веб-фреймворк. https://ru.wikipedia.org/wiki/Веб-фреймворк. (Дата обращения 04.08.2023).

18. Model-View-Controller. https://ru.wikipedia.org/wiki/Model-View-Controller. (Дата обращения 28.08.2023)

19. Spring Security. https://ru.wikipedia.org/wiki/Spring_Security. (Дата обращения 04.08.2023).

20. Запускаем первое веб-приложение на Spring Boot. https://skillbox.ru/media/code/zapuskaem-pervoe-vebprilozhenie-na-spring-boot. (Дата обращения 28.08.2023).

21. Шаблонизатор. https://ru.wikipedia.org/wiki/Шаблонизатор. (Дата обращения 28.08.2023)

22. Persisting Authentication. https://docs.spring.io/spring-security/reference/servlet/authentication/persistence.html. (Дата обращения 28.08.2023)