Н.В. Ульянов

ВИЗУАЛЬНО-ГРАФИЧЕСКИЕ СРЕДСТВА КОНТРОЛЯ ПРОЦЕССОВ НЕФТЕДОБЫЧИ С ИСПОЛЬЗОВАНИЕМ WEB-ТЕХНОЛОГИЙ

Рассмотрена визуализация данных нефтегазодобычи с использованием веб-технологий. Разработан виджет совместного отображения структурной схемы и графика распределения давлений вдоль ствола нефтяной скважины. Рассмотрены вопросы выбора архитектуры подобного решения и средств его реализации.

Визуально-графические системы, web-технологии, нефтегазодобыча

Термины и определения

Web (World Wide Web — глобальная мировая паутина) — система взаимосвязанных гипертекстовыми ссылками документов, доступных через сеть Internet.

HTML (HyperText Markup Language) — язык разметки гипертекста.

Плагин — независимый от основной программы модуль, подключаемый к ней для расширения функциональных возможностей.

Канва (canvas — http:www.w3.org/TR/html-markup/canvas.html) — «холст» для изображения растровой графики в стандарте HTML5 [1].

SVG (Scalable Vector Graphics — http:www.w3.org/Graphics/SVG/) — язык описания векторной графики, основанный на расширяемом языке разметки (XML).

Библиотека-обертка (wrapper library) — дополнительный уровень абстракции между прикладной программой и используемой библиотекой, который использует интерфейсы библиотеки, при этом предоставляя прикладной программе свой, более высокоуровневый интерфейс.

API (Application Programm Interface) — интерфейс, предоставляемый приложением для взаимодействия с ним программными средствами.

Виджет (Widget) — элемент графического интерфейса.

Постановка задачи

Требуется разработать виджет, который на основе данных проекта, текущего моделирования и/или замеров будет в реальном времени отображать структурную схему вертикальной скважины и совмещенный с ней график распределения давлений вдоль ствола (рис. 1). На рисунке введены обозначения для варьируемых величин [2].

Параметры

Глубина забоя — H_Z [м]; глубина подвески насоса — H_N [м]; позиция штуцера подъемника — $H_T = H_N/2$ [м].

Переменные состояния

Давление [МПа] в призабойной зоне — $p_1(t)$, в забое — $p_z(t)$, у приема насоса — $p_3(t)$, на выкиде — $p_5(t)$, в затрубном пространстве верха колонны — $p_4(t)$, на буфере — $p_8(t)$, на контуре питания в пломбе — $p_{пл}$; потери напора на трение в насосно-компрессорной трубе (НКТ) — $\Delta p_t(t) = p_6(t) - p_7(t)$; напор, создаваемый насосом,— $\Delta p_N(t) = p_5(t) - p_3(t)$ и уровень жидкости в затрубном пространстве — $h_4(t)$ [м].

Соответствующие точки графика в координатах <p, H> = <давление [МПа], глубина [м]> для работающей скважины занесены в таблицу варьируемых координат:

Обновляемые (варьируемые) координаты точек виджета

i	0	1	2	3	4	5	6	7	8	9
p	$p_{пл}$	$p_1(t)$	$p_z(t)$	$p_3(t)$	$p_4(t)$	$p_5(t)$	$p_6(t)$	$p_7(t)$	$p_8(t)$	$p_л$
H	H_z	H_z	H_z	H_N	$H_N - h_4(t)$	H_N	H_T	H_T	0	0

Заметим, что десятая точка $i = 10$ соответствует состоянию точки $i = 4$ для выключенной скважины.

В приведенной табл. координаты давлений являются более динамичными по сути решаемых задач, чем координаты уровней, за исключением точки $i = 4$. График снабжается координатной сеткой автоматически регулируемым целочисленным шагом по глубине и давлению в зависимости от исходных настроек $<H_z, P_{пл}, P_л>$.

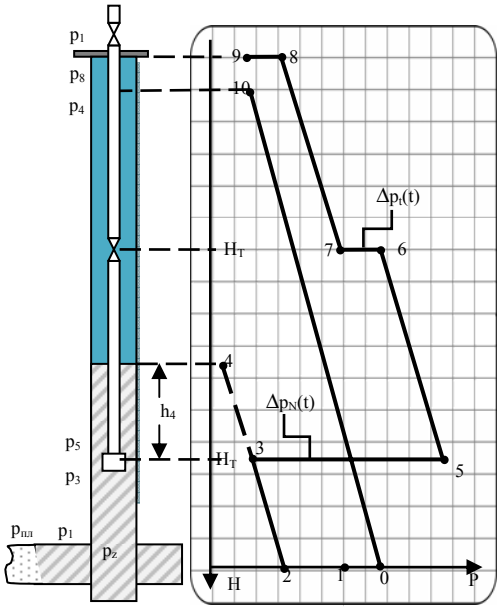

Рис. 1. Схема скважины и график распределения давлений

Архитектура проекта

Рассмотрим основные вопросы реализации подобного рода программного обеспечения.

Первый вопрос — архитектура проекта как сетевого приложения.

Можно выделить три основных вида архитектур сетевых приложений [3].

1. Файл-серверная архитектура (рис. 2).

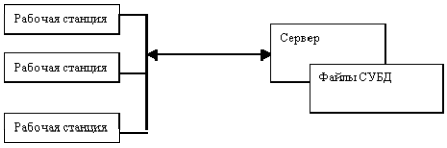

Рис. 2. Файл-серверная архитектура

При этом способе организации основной функцией сервера является хранение данных в виде одного или нескольких файлов, а вся обработка выполняется клиентской станцией. Для получения из базы даже одного значения передается весь файл, а при записи значения он еще и целиком блокируется, что при большом числе пользователей создает повышенную нагрузку на сеть и проблемы доступа к ресурсу на запись.

2. Клиент-серверная архитектура с «толстым» клиентом (рис. 3).

Описанные выше недостатки файл-серверной системы устраняются за счет того, что обработка происходит на самом сервере: сделанный клиентом запрос отсылается к базе, а клиент получает лишь интересующий его ответ. При этом большая часть вычислений так же находится на рабочей станции. Однако администрировать такую систему по-прежнему неудобно, так как ее обновление требует обновления ПО на каждой станции.

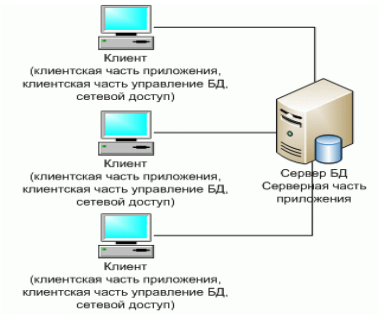

Рис. 3. Клиент-серверная архитектура

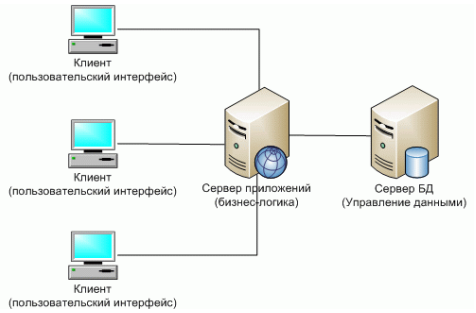

Рис. 4. Трехзвенная архитектура

3. Клиент-серверная архитектура с «тонким» клиентом, трехзвенная (рис. 4).

Не только получение данных, но и вся бизнес-логика выносится на так называемый сервер приложений, на рабочей же станции остается лишь интерфейс для обращения к нему.

При этом упрощается администрирование, так как клиентское ПО в администрировании практически не нуждается, и обеспечивается высокий уровень безопасности за счет того, что клиент не имеет непосредственного доступа к базе, в которой в общем случае могут быть данные, для конечного пользователя этой системы не предназначенные. Все обращения происходят через дополнительный уровень, при этом клиент не имеет представления об организации данных на сервере. По вышеназванным причинам трехзвенная архитектура стала де-факто стандартом в системах с большим числом пользователей и ее использование здесь будет оправдано.

Классическим и наиболее распространенным примером трехзвенной архитектуры являются web-приложения. В дальнейшем будем рассматривать именно их.

Второй вопрос, возникающий при конструировании подобного рода решений, касается выбора узла, со стороны которого целесообразнее формировать изображения.

Здесь уместно рассмотреть два подхода:

1. Генерация графиков на стороне backend'а (сервера). Этот способ может успешно применяться для статических изображений (а также динамических с фиксированным порядком фреймов) по той причине, что серверные языки, как правило, обеспечивают значительное удобство при проведении различного рода вычислений. Но такой способ реализации делает практи-

чески невозможной интерактивность изображения, т.е. его реакцию на действия пользователя, так как требует слишком частых запросов на достаточно большие объемы данных.

2. Генерация на стороне frontend'a (рабочей станции). В этом случае с сервера на рабочую станцию по запросу передаются массивы данных, на основе которых клиент формирует изображение. Это нивелирует недостаток предыдущего способа. Таким образом реакция на стороне клиента предоставляет больший спектр возможностей.

Третий вопрос — выбор конкретной технологии для получения изображения с учетом решений по предыдущим вопросам.

Основные варианты следующие.

1. Использование сторонних плагинов (plugin) к браузеру, например Adobe Flash, Silverlight. Этот путь обусловливает зависимость от установки и включенности этих плагинов у пользователя, хотя их наличие и весьма вероятно, но отнюдь не гарантировано. Кроме того, разработка на Adobe Flash требует приобретения лицензии.

2. Использование векторной графики, описанной в SVG-формате. Так как SVG является подмножеством XML, для манипуляций над ним подходят любые средства, предназначенные для XML, однако особенности синтаксиса могут привести к большему объему кода при описании изображения.

3. Использование канвы. Объект, пришедший на замену сторонним плагинам и SVG, появившийся в HTML совсем недавно и потому имеющий значительные проблемы с поддержкой в старых браузерах вплоть до полной неработоспособности.

Второй и третий варианты сводят манипулирование изображением к коду на javascript, являющемся абсолютным стандартом в области программирования на клиентской стороне в web, первый же требует значительно большего числа дополнительных знаний, поэтому рассматривать его в дальнейшем не будем.

И, наконец, ***четвертый вопрос***: использовать ли готовые библиотеки-обертки над API, предоставляемым канвой, или писать свою для конкретных нужд. Первый способ характеризуется меньшими трудозатрами, но потенциально может сильно ограничить в гибкости. Второй же, напротив, предоставляет возможность значительно большего контроля над кодом, но в то же время зачастую позволяет программисту совершать ошибки и соответственно влечет высокие трудозатраты.

Алгоритм функционирования

Схема данных

Рассмотрим обобщенный алгоритм функционирования разрабатываемого виджета.

Рассматриваемый виджет W состоит из двух частей — структурной схемы скважины S и графика распределения давлений G:

W = (S, G).

При этом в структурной схеме скважины имеет смысл выделить грунт T и динамическую часть схемы D – S = (T, D).

График распределения давлений G состоит из сетки Q, множества линий графика L, осей xx, yy и рамки вокруг графика b.

G = (Q, L, B)

При этом сетка характеризуется горизонтальным и вертикальным шагом линий на ней (hx и hy соответственно) и обрамляющего прямоугольника B, задающего отображаемую часть системы координат, и включает в себя коор-

динатные оси xx и yy. Обрамляющий прямоугольник будем задавать при помощи двух ограничивающих его точек p1 и p2:

Q = (hx, hy, xx, yy, B),

B = (p1, p2).

Каждая линия графика характеризуется двумя точками, составляющими ее:

L = (L1, L2, L3, ... Ln),

Li = (p1, p2).

Причем точку можно рассматривать как вырожденный случай прямой при p1 = p2.

Уровни темпоральности

В данном случае по частоте обновления объектов можно выделить три уровня темпоральности: T1, T2, T3.

К первому, низшему уровню, объекты которого рисуются лишь однажды при инициализации виджета и остаются неизменными на протяжении всего времени функционирования виджета, относятся грунт, обрамляющая рамка сетки и оси координат.

T1-> {T, b, xx, yy}

Сетку отображения пересчитывают в двух случаях:

1) если линии не помещаются в отображаемую часть графика;

2) если минимальная область графика, включающая в себя все существующие на нем линии, занимает менее 50 % от всего графика.

Для этой проверки введем предикат NR(L, Q), истинный в случае, когда выполняется одно из вышеназванных условий.

Таким образом, сетка будет представлять собой второй уровень темпоральности:

T2 -> {Q}.

К высшему уровню будем относить объекты, перерисовываемые при получении очередной порции данных о точках. Таковыми являются линии и динамическая часть структурной схемы.

T3-> (D, L)

Алгоритм

При получении данных от источника необходимо преобразовать этот массив данных data в список линий L.

lines = lines(data) -> L.

У нас в распоряжении имеется несколько систем координат. Первая относится к рассматриваемой предметной области, где по одной оси располагается глубина скважины (направление оси — сверху вниз), по другой — давление (слева направо). Это система координат, в которой мы изначально получаем данные. Вторая — система координат HTML-элемента, используемого в качестве холста для программирования. Традиционно в средствах отображения информации принят отсчет координат от левого верхнего угла. Задание координат в CSS возможно с использованием различных единиц измерения (дюймы, сантиметры, в процентах от обрамляющего блока и т.д. [4]). Выберем в качестве единиц измерения экранные дюймы. Третья — система координат рисунка структурной схемы, размеры элементов которой удобнее всего измерять в процентах от высоты всего рисунка, что приводит к необходимости введения еще двух преобразующих операторов:

natural_to_inch = natural_to_inch(L) -> L_g, где data_g — массив линий с координатами, выраженными в дюймах и пригодными для отображения на нашем «холсте».

natural_to_percent = natural_to_percent(L) -> L_h, где L_h — вертикальные размеры элементов структурной схемы скважины в процентах от ее высоты.

recalculate = recalculate(greed) -> greed1, где greed1 — сетка, обновленная таким образом, чтобы на ней умещались и все объекты, но при этом не было лишних пустот.

Введем следующие вспомогательные функции:

draw = draw(item) — нарисовать объект.

fetch_data = fetch_data() — (получить порцию данных).

Теперь запишем алгоритм функционирования виджета на псевдокоде:

```
draw(T1) //Отобразим элементы первого уровня темпоральности
for data in fetch_data(): //получим новую порцию данных
L = lines(data)
if NR(our_lines, Q): //если необходимо пересчитать сетку
Q = recalculate(Q)//пересчитаем сетку
draw(T2) //отобразим элементы второго уровня темпоральности
LL = natural_to_inch(L) //получим линии для графика
items = natural_to_percent(L) //получим элементы для структурной схемы
draw(T3)//Отобразим элементы третьего уровня темпоральности
```

Вывод

По результатам проведенной работы был спроектирован и разработан искомый виджет, отображающий данные замеров в виде совмещенных структурной схемы скважины и графика распределения давлений вдоль ствола.

ЛИТЕРАТУРА

1. *Canvas* — canvas for dynamic graphics [Электрон. ресурс]. Режим доступа: http://www.w3.org/TR/html-markup/canvas.html.

2. *Ведерникова Ю.А., Соловьев И.Г.* Разработка и использование гидродинамических моделей скважинных систем, оборудованных установками погружных электроцентробежных насосов // Вестн. кибернетики. Тюмень: Изд-во ИПОС СО РАН, 2002. Вып. 1. С. 85–91.

3. *Рябов В.А., Несвижский А.И.* Архитектурные особенности проектирования и разработки веб-приложений [Электрон. ресурс]. Режим доступа: http://www.intuit.ru/department/internet/mwebtech/5.

4. *Bert Bos.* Web Style Sheets. CSS tips and tricks [Электрон. ресурс]. Режим доступа: http://www.w3.org/Style/Examples/007/units.en.html.

N.V. Ulyanov

Visual-and-graphical tools for oil production control using web-technologies

The article considers data visualization in oil and gas production using web-technologies. Subject to development being a widget of joint display regarding a structural scheme and a graph of pressure distribution alongside a wellbore. The author considers questions of selecting the architecture of such solution and its implementation tools.

Visual-and-graphical systems, web-technologies, oil and gas production.