УДК 004.82
DOI 10.25205/1818-7900-2018-16-4-115-126

П. В. Мызников

Новосибирский государственный университет
ул. Пирогова, 1, Новосибирск, 630009, Россия

miznikov72@gmail.com

РАЗРАБОТКА ПРЕЦЕДЕНТНО-ОРИЕНТИРОВАННОГО ПОДХОДА ОБРАТНОГО РЕИНЖИНИРИНГА WEB-ИНТЕРФЕЙСОВ

Статья посвящена применению рассуждений на основе прецедентов (case-based reasoning, CBR) в области web-разработки. Учитывая собственный практический опыт в данной сфере, автор предлагает автоматизировать составление html/css-макета путём агрегирования прецедентов кода из предыдущих опытов, что на методологическом уровне крайне близко к подходу CBR, а именно: предлагается методика конструирования систем, генерирующих html-код из растрового изображения и основанных на подходе рассуждений на основе прецедентов. В частности, в статье изложена оригинальная теория составления структуры изображения и описан алгоритм получения такой структуры. Кроме того, рассматривается модификация подхода case-based reasoning, которая позволяет получить требуемый результат решения поставленной задачи. Также описываются результаты прохождения экспериментов разработанной системы в определённых условиях.

Ключевые слова: рассуждения на основе прецедентов, компьютерное зрение, оптическое распознавание символов, обратный инжиниринг, web-разработка.

Введение

Web-технологии являются одним из самых развитых направлений современных информационных технологий. Они используются не только для разработки коммерческих сайтов компаний, но и как один из важных узлов любой ИТ-инфраструктуры. Соответственно можно говорить о том, что практически вся отрасль информационных технологий в том или ином виде использует web-разработку.

Вместе с тем автоматизация самой web-разработки является недостаточно полной. Существуют средства автоматизации развёртывания сред окружения, механизмы конструирования каркасов приложений и некоторые другие направления. Однако почти не охваченным остаётся одна из ключевых задач: создание HTML/CSS-макетов. С одной стороны, эта часть считается одной из наиболее рутинных; с другой – ряд особенностей этого процесса не позволяет применить классические средства автоматизации. К таким особенностям можно отнести: наличие не формализуемых требований к макету, вариативность корпоративных стандартов написания кода, требование кроссбраузерности и кроссплатформенности, что предполагает постоянную адаптацию решения к новым средам.

Автоматизация создания HTML/CSS-макетов не только ускорит процесс разработки web-приложений, но и сделает его более гибким, позволяя масштабировать его результаты, проверять гипотезы, помогать тестированию. Это обусловливает целесообразность описываемой разработки.

Ввиду сказанного объектом исследования является обратный реинжиниринг в области web-технологий, а предметом – автоматизация обратного реинжиниринга интерфейсов web-

Мызников П. В. Разработка прецедентно-ориентированного подхода обратного реинжиниринга web-интерфейсов // Вестн. НГУ. Серия: Информационные технологии. 2018. Т. 16, № 4. С. 115–126.

© П. В. Мызников, 2018

приложения. Цель работы – создать технологию построения систем генерации html-кода на основе растрового изображения.

Методическую основу теоретических оснований составляет подход *case-based reasoning* (CBR, рассуждения на основе прецедентов). Кратко описывая его суть, можно сказать, что это способ решения проблем путём адаптации решений аналогичных проблем в прошлом к текущей ситуации. Такой подход выбран в связи с поставленной в работе гипотезой о том, что формализация изображений (чем, по сути, является создание HTML-макета) посредством трансдуктивных выводов, или рассуждений по аналогии, порождает результат, наиболее близкий к тому, что производит человек. К тому же такая схема позволяет решить описанные выше проблемы автоматизации.

Следует отметить, что поставленная выше гипотеза обусловливает использование подхода, который отличает данное исследование от смежных (см. раздел «Обзор литературы»), что можно рассматривать как элемент научной новизны. Кроме того, к новым результатам можно отнести разработку алгоритма извлечения структуры элементов изображения.

Обзор литературы

В последние годы в зарубежной практике появляется повышенный интерес к поставленной проблеме.

Полученный в работе [1] результат (проект DeepCoder) даёт основания полагать, что современный уровень развития искусственного интеллекта позволяет автоматизировать создание артефактов работы ИТ-специалистов, которые ранее считались не поддающимися автоматизации. Так, DeepCoder – это система генерирования программного кода на основании входных и выходных данных требуемого алгоритма. Хотя такая проблематика отличается от рассматриваемой в текущей работе, разработки, полученные в ходе этого проекта, послужили основой для многих других исследований [2; 3], родственных рассматриваемой задаче. К таким разработкам можно отнести, в частности, использование моделей машинного обучения, основанных на рекуррентных нейронных сетях, и специальные техники поиска блоков кода.

Разработку достаточно близкой задачи осуществлял Т. Нгуен [4]: обратный инжиниринг интерфейсов мобильного приложения. Напрямую перенести этот опыт на инжиниринг web-интерфейсов не получится. Однако сильная сторона исследования – техники оптического распознавания символов (OCR) – может быть повторно использована для распознавания элементов изображения страницы Интернет-сайта.

Самым последним успешным случаем решения описываемой задачи можно назвать работу организации UIzard Technologies [5] под названием «pix2code». Авторы отмечают схожесть данной задачи с задачей генерирования текста на естественном языке, описывающего изображение. Соответственно, и метод они выбирают схожий: в основе их решения лежит каскад рекуррентных нейронных сетей типа LSTM (long-short term memory, долгая краткосрочная память). При несомненных достоинствах этого подхода и успешных практических результатах следует выделить недостатки.

Наиболее значимым видится, как утверждают сами авторы, отсутствие использования экспертной эвристики. Принимая преимущества этой особенности в некоторых условиях, необходимо отметить, что в общем случае это ведёт к ограниченности подхода, так как многие требования к реализации макетов нельзя извлечь исключительно из изображения: необходим контекст, который существует в знаниях экспертов.

Другой важный недостаток – формирование обучающей выборки. Нейронные сети, несомненно, мощный инструмент машинного обучения, однако он требует на вход огромное количество данных (порядка миллионов записей), чтобы быть по-настоящему эффективным. Безусловно, учитывая колоссальные размеры Интернета, такую выборку собрать технически несложно. Но возникает вопрос о репрезентативности данных: далеко не всякий web-сайт положительно скажется на обучении нейронных сетей. В связи с этим появляется дополнительная задача определения качества HTML-кода, что видится существенным увеличением сложности реализации.

В работах [6; 7] рассуждения на основе прецедентов и методы онтологического моделирования применялись для диагностирования заболеваний позвоночника. Знания извлекались и обрабатывались при помощи онтологической модели данной предметной области [8]. В качестве прецедентов рассматривались истории болезни пациентов. Множество прецедентов формально было представлено в виде прецедентной модели, являющейся частным случаем булевозначной модели [9]. Порождение оценочных знаний о болезнях пациентов производилось при помощи нечётких моделей, полученных фазификацией булевозначных прецедентных моделей [9; 10].

Опыт и результаты этих исследований, учёт их сильных и слабых сторон будут использованы в текущей работе.

Постановка задачи

Опишем подробнее, что требуется от системы. Алгоритм должен распознать структуру изображения, выделить как можно больше характеристик объектов в этой структуре (цвет, шрифт, отступы…) и сгенерировать код, реализующий построение распознанной структуры.

При этом не требуется:

- распознать **все** атрибуты,
- воссоздать картинку пиксель-в-пиксель.

В идеале достаточно, чтобы алгоритм безошибочно сгенерировал около 85 % кода так, чтобы HTML-специалист любой квалификации мог без труда добавить оставшиеся 15 %, при этом чтобы не нужно было принципиально менять структуру кода, а также думать о решении стандартных задач. Поставим этот показатель в качестве ориентира при разработке прототипа.

Во введении и обзоре литературы были упомянуты условия, которые следует учесть при решении задачи. Необходимо разработать ряд требований к системе таких, чтобы можно было получить положительные результаты при работе в этих условиях.

- **Условие 1. Наличие не формализуемых требований к макету.** Изображение само по себе не несёт всю полноту информации о том, каким должен быть конечный HTML-макет. Соответственно, требуется, чтобы в разрабатываемую систему можно было заносить дополнительное знание о требованиях.
- **Условие 2. Вариативность корпоративных стандартов написания кода.** Существует несколько методологий HTML-вёрстки (адаптивная – не адаптивная, bem-методология, bootstrap и т. д.). Более того, как правило, в каждой команде разработчиков существуют внутренние стандарты. Соответственно, от системы требуется умение создавать код в разных стилях и стандартах.
- **Условие 3. Требование кроссбраузерности и кроссплатформенности.** HTML-код должен не только быть валидным, но и одинаково восприниматься всеми современными браузерами и операционными системами.

Кроме того, необходимо ввести ограничения к входным данным на этапе разработки прототипа. Изображение сайта не должно содержать:

- градиентный тип заливки,
- элементы анимации,
- «выпадающие элементы» (элементы, возникающие на странице в результате определённых событий).

Данные ограничения вводятся на этапе создания прототипа в целях лучшего решения базовых проблем. В дальнейшем предполагается создание алгоритма, работающего и без этих условий.

Архитектура

В основе архитектуры лежит попытка воссоздать процесс составления кода человеком. Базовое утверждение состоит в том, что человек, формализуя визуальную информацию, преимущественно применяет трансдуктивные рассуждения: он извлекает из изображения структуры разного уровня абстракции и описывает эти структуры по аналогии со своим или чужим прошлым опытом.

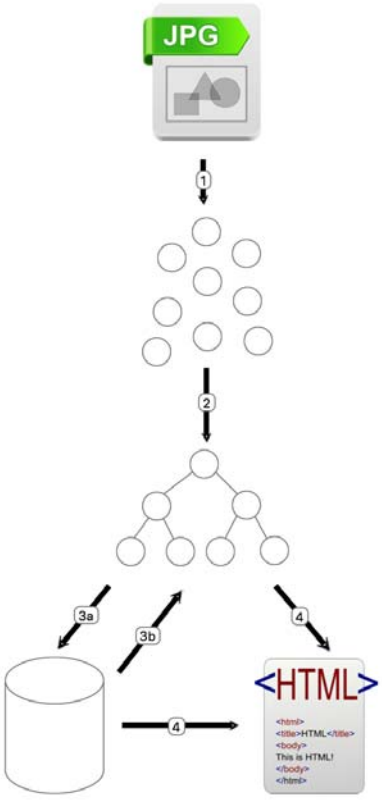

Рис. 1. Схема генерации HTML-кода из изображения

В частности, такая ситуация наблюдается при написании html-кода. Встречая незнакомый паттерн расположения блоков, специалист находит в справочниках и вообще в литературе, как реализовать это на языке html. Далее, с опытом, обращений к справочникам становится всё меньше, но принцип остаётся тем же: сталкиваясь с очередным шаблоном, он достаёт из памяти подходящий пример и адаптирует к текущей ситуации.

Данный процесс реализуется следующим образом (рис. 1):

1) из изображения извлекаются все элементы (текст, картинки, блоки и т. п.);

2) извлечённые элементы собираются в древовидную структуру согласно специальному алгоритму;

3) совершается префиксный обход дерева, на каждой итерации которого происходит обращение к базе прецедентов с целью поиска подходящего описания архитектурного паттерна, лежащего в обрабатываемом узле, на языках HTML и CSS;

4) полученные на предыдущем шаге артефакты сохраняются в виде требуемых файлов.

Шаги 1 и 4 являются инженерными задачами, которые уже имеют решения в общедоступной практике. Шаги 2 и 3 – наукоёмкая часть архитектуры. В разделах «Извлечение структуры изображения» и «Построение CBR-системы» приводится разработка теории и алгоритмов, которые были использованы для реализации этих шагов.

Извлечение структуры изображения

Цель данного этапа – получить так называемую **структуру** растрового изображения, т. е. математический объект заданного формата, описывающий взаимное расположение частей изображения. Поставленная задача является задачей построения дерева. В данной статье предлагается жадный тип алгоритма решения.

На вход алгоритму подаётся множество узлов, каждый из которых содержит ровно один элемент изображения, упорядоченное по координатам левой верхней точки описанных прямоугольников.

Общая идея заключается в последовательной обработке элементов данного множества и попытке на каждой итерации объединить обрабатываемый элемент со следующим в определённый узел (рис. 2). При объединении элементов в узлы мощность множества будет сокращаться. Алгоритм завершит работу, когда во множестве останется ровно один элемент – именно этот элемент, представляющий собой дерево всех элементов, является решением задачи.

Проведём оценку сложности алгоритма по времени и по памяти.

Заметим, что алгоритм не использует дополнительную память, т. е. не зависит от количества элементов в множестве. Соответственно, оценка по памяти равна $O(1)$.

С оценкой по времени ситуация сложнее. В зависимости от расположения элементов, алгоритм может обладать совершенно разной сложностью.

В лучшем случае изображение состоит из ряда элементов, расположенных в линию, горизонтальную или вертикальную. Тогда алгоритму достаточно пройти один раз по множеству и объединить их все в один определённый узел. Оценка сложности по времени в этом случае

линейная – $O(n)$. К сожалению, такой случай почти не встречается на практике (только, быть может, в очень узких задачах).

Чтобы оценить худший случай, проанализируем, как собирается дерево. При каждом проходе элементов (см. рис. 2, цикл на строке 2) собираются узлы, которые в конечном дереве будут лежать на одном уровне. При этом каждый такой проход в худшем случае оценивается как $O(n)$. Соответственно, оценка сложности всего алгоритма равна $O(n*h)$, где h – высота дерева. Наихудшая высота дерева равна количеству элементов: на каждом проходе цикла получается максимум один узел. Получаем $O(n^2)$ как оценку времени выполнения алгоритма в худшем случае. Хорошая новость состоит в том, что таких примеров на практике хоть и больше, чем наилучших случаев, но всё же не так много.

Require: $nodes$ – множество элементов, упорядоченных по абсциссе и ординате

```
1:  while |nodes| > 1 do
2:      for orientation ← [HOR,VERT] do
3:          i ← 0
4:          while i < |nodes| do
5:              suitableNode ← findSuitableNode(nodes_i, orientation)
6:              if suitableNode is not null then
7:                  if node_i is composite then
8:                      nodes_i.addChild(suitableNode)
9:                  else
10:                     newNode ← Node()
11:                     newNode.addChild(nodes_i)
12:                     newNode.addChild(suitableNode)
13:                     newNode.orientation = orientation
14:                     nodes_i ← newNode
15:                 delete suitableNode
16:                 i ← i + 1
```

Ensure: $|nodes| == 1$ и $nodes_1$ – корневой узел дерева, содержащий все элементы из исходного множества

Рис. 2. Алгоритм построения иерархии элементов изображения

В среднем ожидается, что на каждой итерации цикла (см. рис. 2, строка 2) количество элементов будет логарифмически сокращаться, поэтому среднюю сложность можно оценить как $O(n\log n)$. Для более точных данных об истинном распределении наилучших, наихудших и прочих случаев, а также справедливости приведённых в данном параграфе формул необходимо провести дополнительные эксперименты и исследования.

Рассуждения на основе прецедентов

Рассуждения на основе прецедентов (CBR, case based reasoning) – одно из успешных направлений развития искусственного интеллекта. Будучи изобретённым в конце 70-х гг. XX века, case-based reasoning использовался в разработке многих интеллектуальных систем.

Работы Шэнка и Абельсона 1977 г. [11] считаются истоками рассуждений на основе прецедентов. Роджер Шенк предложил следующую концепцию: наше знание о мире организовано в виде пакетов памяти, хранящих эпизоды жизни. Такие пакеты (MOPs – memory organizations packets) и их элементы не изолированы, а пересекаются с нашими ожиданиями развития событий (сценариями). В свою очередь, MOPs образуют иерархию, где более общие

пакеты объединяют более специфичные. Если МОР содержит ситуацию, в которой некоторая проблема была успешно решена, и человек находит себя в подобной ситуации, то он стремится использовать предыдущий опыт, чтобы найти решение. Таким образом, вместо следования общему набору правил происходит повторное применение схемы решения в новых, но схожих условиях.

В 1990-х гг. было написано большое количество научных работ, посвящённых этому направлению [12–17]. Тем не менее в 2000-х гг. популярность CBR стала падать вследствие вытеснения другими подходами. Несмотря на это, рассуждения на основе прецедентов остаются достаточно эффективной методологией, которую до сих пор используют в самых разных предметных областях.

Кратко описывая суть рассуждений на основе прецедентов, можно сказать, что это способ решения проблем путём адаптации решения схожих проблем в прошлом к текущей ситуации. Процесс нахождения такого решения называется CBR-циклом и состоит из 4 фаз [14].

5) **Извлечение.** Из хранилища прецедентов извлекается прецедент с проблемой, наиболее похожей на ту проблему, которую нужно решить.

6) **Адаптация.** Решение извлечённого прецедента адаптируется так, чтобы решить исходную проблему.

7) **Оценка.** Полученное решение оценивается: если оно не удовлетворяет заданным критериям, то либо извлекаются дополнительные прецеденты и снова происходит попытка адаптации, либо процедура завершается с неудачным результатом.

8) **Сохранение.** Прецедент с полученным решением сохраняется в хранилище прецедентов.

Более подробно остановимся на представлении прецедентов. В CBR прецедент – это тройка элементов:

- проблема – состояние мира / предметной области, при котором возникает прецедент,
- решение – приемлемый в данном контексте ответ для проблемы,
- результат – состояние мира / предметной области после применения решения.

Построение CBR-системы генерации кода

Как было сказано в предыдущем разделе, чтобы построить CBR-систему, необходимо определить следующие вещи: формат прецедента, меру сходства, процедуру адаптации, словарь, а также реализацию 4 шагов CBR-цикла.

В разделе «Архитектура» сказано, что генерация кода производится путём префиксного обхода дерева – структуры изображения. На каждом этапе этого обхода запускается CBR-цикл. Требуется определить, что является проблемой, решением и результатом в контексте решаемой задачи. Иными словами, описать структуру прецедента.

- **Проблема**: набор свойств обрабатываемого узла и свойств его прямых потомков.
- **Решение**: шаблоны html- и css-кода, реализующие вёрстку описываемой структуры.
- **Результат**: html/css-код, сгенерированный из решения-шаблона применительно к конкретному случаю.

Так, при обработке каждого узла дерева формируется <<проблема>>: описание текущего элемента структуры изображения вместе с описанием структуры изображения на один уровень детализации глубже. В хранилище лежат прецеденты, проблемы которых представляют собой типовые случаи расположения элементов, а решения – типовые варианты их вёрстки. Задача CBR-цикла – используя хранимые решения, получить html/css-код для реализации текущей структуры. Опишем этот цикл подробнее. Для условных обозначений назовём обрабатываемый прецедент общепринятым термином *новый прецедент*. Дальнейшие термины будем вводить по мере их появления.

Извлечение. *Новый прецедент* содержит проблему, но не имеет решения и результата. Задача данного этапа – извлечь из хранилища прецеденты, максимально близкие по проблеме к *новому прецеденту*. В общем случае имеем задачу n-классовой классификации, где n – количество прецедентов в хранилище. На этапе построения концепции выберем в качестве решения метод k ближайших соседей с евклидовой мерой расстояния. Для этого векторизуем и нормализуем проблемы. Это означает, что, во-первых, все категориальные признаки при-

ведём к числовому виду; во-вторых, заменим абсолютные значения относительными по формуле $x_i = \frac{x_i}{x_{\max} - x_{\min}}$. Далее, вычисляем расстояния между текущим вектором и векторами прецедентов в хранилище $d(x, y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$ и выбираем прецедент, расстояние до которого оказалось минимальным. В терминах CBR такой прецедент называется *извлечённым*.

Адаптация. Шаг адаптации реализуется алгоритмом, принимающим на вход проблему *нового* прецедента и решение *извлечённого*, а отдающим на выход результат. Прецедент, составленный из проблемы *нового* прецедента, решения *извлечённого* прецедента и результата, полученного на шаге адаптации, называется *решённым*, а само решение называется *предлагаемым*.

В текущем исследовании предлагается деривационный тип адаптации, т. е. перегенерация старого решения в новых условиях. Так как решение прецедента – это шаблон кода, то результат получается путём применения выбранного языка шаблонизации. Трудность заключается в том, что результаты разных прецедентов в ходе обработки структуры могут конфликтовать друг с другом. Особенно это касается конфликтов в css-коде: ситуации, когда различные результаты описывают по-разному один и тот же css-класс. Поэтому на этапе адаптации важным является процесс разрешения конфликтов.

Оценка. В классическом подходе CBR этап оценки служит валидацией полученного результата. Прецедент, который положительно прошёл данный этап, называется *протестированным*, а его решение – *подтверждённым*. Что касается текущей работы, задача оценки качества html/css-кода заслуживает отдельного исследования. На этапе проверки концепции данный шаг опускается, а проверка полученного решения осуществляется для всего документа целиком, а не на каждой итерации CBR-цикла, как предлагается в методологии.

Сохранение. Последний шаг цикла призван сохранить подтверждённое решение в базе, для того чтобы использовать его в будущем для решения схожих проблем. В текущей работе этот шаг несколько отличается от классического подхода. В отличие от использования одного хранилища прецедентов разрабатываемая CBR-система поддерживает «глобальное» хранилище и несколько «локальных» хранилищ (см. таблицу). Первый тип – это база прецедентов, используемая по умолчанию для всех изображений. Второй – база прецедентов, используемая только в рамках обработки одного документа. Можно сказать, что «локальное» хранилище – это контекст документа: оно содержит результаты решённых проблем в ходе его обработки.

Отличия локального и глобального хранилищ

Локальное хранилище	**Глобальное хранилище**
Хранится в оперативной памяти	Хранится во внешней памяти
Уничтожается после обработки документа	Существует независимо от обработки документа
Заполняется в процессе обработки документа	Заполняется специальной процедурой до обработки документа
Прецеденты имеют приоритет при извлечении	

Смысл состоит в том, что на этапе сохранения решённый прецедент всегда записывается только в локальное хранилище. При этом на этапе извлечения поиск происходит во всех базах, но преимущество есть у прецедентов в локальном хранилище.

Другими словами, глобальное хранилище служит источником прецедентов, имеющих решения генерации кода типичных примеров расположения элементов. Далее локальное хранилище используется для точной и быстрой адаптации этих решений в контексте конкретного документа.

Результаты и дальнейшая работа

Компьютерная система, реализующая описанный выше подход, была протестирована на множестве изображений существующих сайтов. Такой вид тестирования позволил оценить, насколько близко к человеческому решению получает результат построенный алгоритм. В качестве примера рассмотрим одну из страниц сайта Новосибирского Государственного Университета.

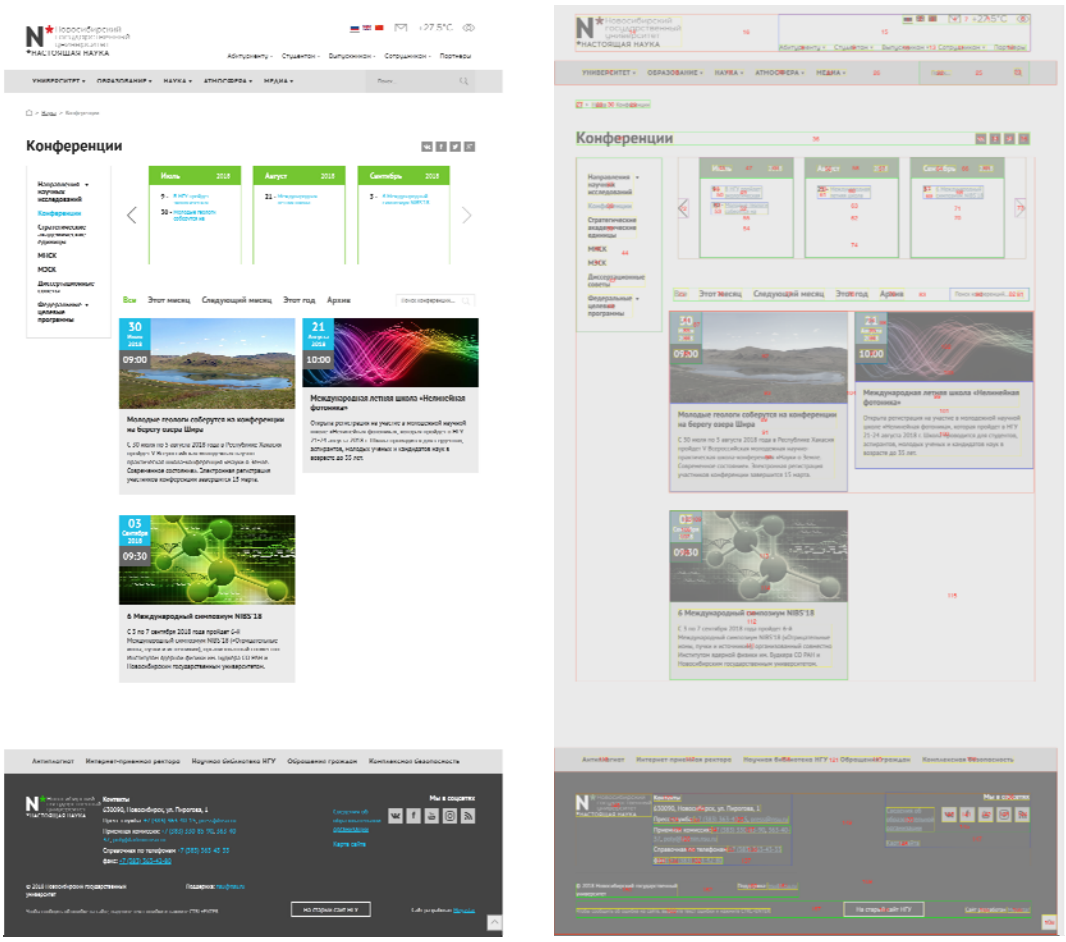

Рис. 3. Оригинал обрабатываемого изображения (слева) и выделенные элементы и узлы (справа). Узлы обведены цветными прямоугольниками и пронумерованы

В результате выполнения первого этапа обработки изображения были выделены элементы и сгруппированы в узлы (рис. 3). Далее, описанный выше алгоритм сформировал из полученных узлов древовидную структуру (рис. 4). Полученный результат имеет достаточно хорошее качество (рис. 5). Несмотря на некоторые ошибки (не был выбран подходящий шрифт, не подгружены некоторые изображения, небольшие ошибки в размерах), система справилась с самой главной задачей: воссоздала структуру элементов и сгенерировала легко поддерживаемый человеком код.

Из данного примера видно, что прототип системы удовлетворяет поставленным требованиям. В общем, учитывая эксперименты на других изображениях, можно утверждать, что средняя точность работы системы составляет 87 %. Также стоит заметить, что увеличение структурной сложности (добавление колонок, «резиновой» ширины) ожидаемо снижает результаты. Эти выводы полезны для определения следующих направлений развития.

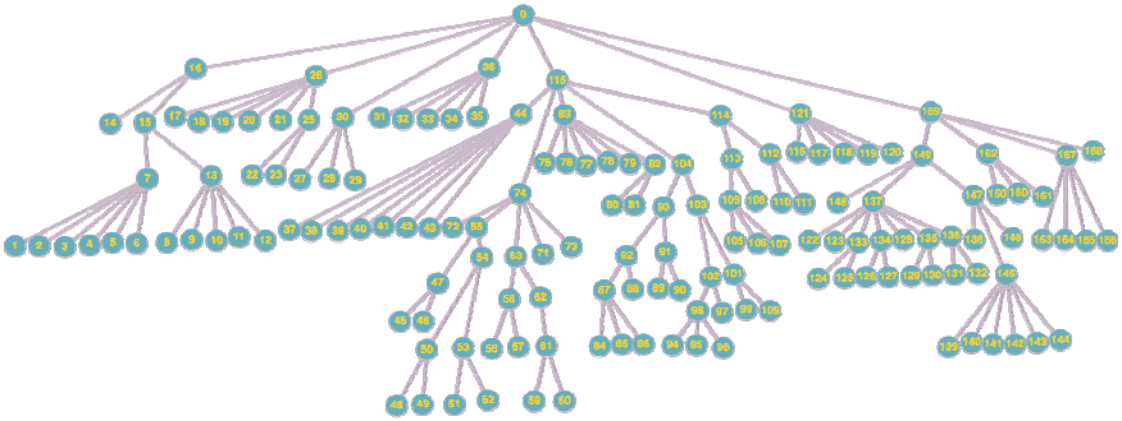

Рис. 4. Структура элементов обрабатываемого изображения.
Числа внутри узлов соответствуют индексам меток, расставленных на рис. 3

Помимо количественных требований, в постановке задачи были выдвинуты три условия, в которых система должна эффективно функционировать: 1) наличие не формализуемых требований к макету, 2) вариативность корпоративных стандартов написания кода, 3) требование к кроссбраузерности и кроссплатформенности. Выполнение всех этих условий учитывается в системе за счёт вариативности заполнения хранилища прецедентов. Действительно, помимо автоматического получения множества прецедентов, есть возможность создавать их с помощью экспертов. В этом случае в решениях-шаблонах будет находиться экспертный опыт и дополнительное знание. А производя замену хранилищ, каждое из которых специфицировано для разных стилей и стандартов, можно получать заданный результат в разном исполнении.

Следующими шагами в развитии разработанной концепции являются:

1) разработка процедуры автоматического заполнения хранилища прецедентов;

2) сбор тренировочного набора реальных примеров изображений и проведение экспериментов на этом наборе;

3) разработка наилучшей меры близости прецедентов;

4) разработка этапа оценки в CBR-цикле.

Таким образом, получен прототип интеллектуальной системы, эффективность которой доказана экспериментально. Предлагаемый подход является перспективным решением многих практических проблем в web-разработке. Важно и то, что разработанная теоретическая модель распознавания структур изображений может быть обобщена и использована в других вопросах компьютерного зрения.

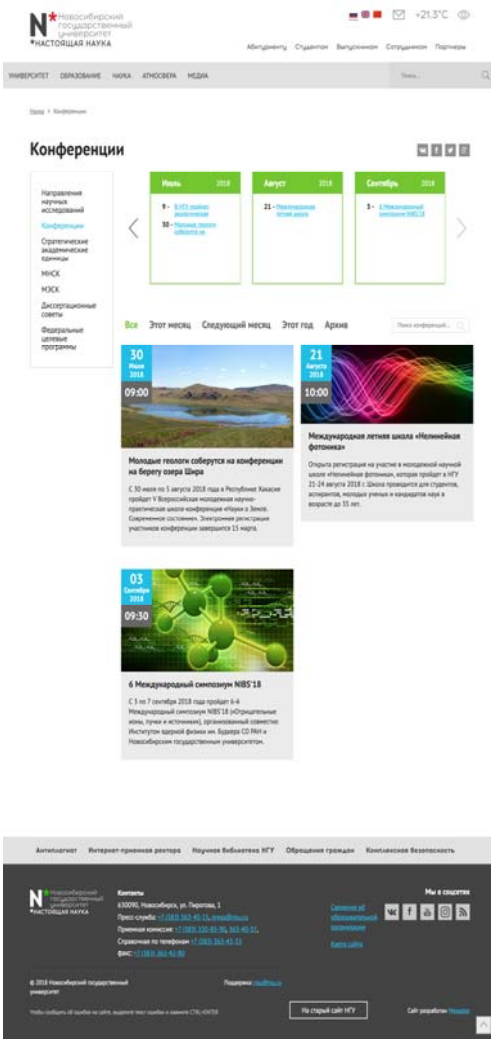

Рис. 5. Изображение web-страницы, сгенерированной системой

Список литературы

1. DeepCoder: Learning to Write Programs / M. Balog [et al.]. 2016. Nov. arXiv: 1611.01989. URL: http://arxiv.org/abs/1611.01989.

2. TerpreT: A Probabilistic Programming Language for Program Induction / A. L. Gaunt [et al.]. 2016. Aug. arXiv: 1608.04428. URL: http://arxiv.org/abs/1608.04428.

3. Latent Predictor Networks for Code Generation / W. Ling [et al.]. 2016. Mar. arXiv: 1603.06744. URL: http://arxiv.org/abs/1603.06744.

4. *Nguyen T. A., Csallner C.* Reverse Engineering Mobile Application User Interfaces with REMAUI (T) // 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE). Nov. 2015. P. 248–259. DOI 10.1109/ASE.2015.32.

5. *Beltramelli T.* pix2code: Generating Code from a Graphical User Interface Screenshot. 2017. arXiv: 1705. 07962. URL: http ://arxiv.org/abs/1705.07962.

6. *Palchunov D., Yakhyaeva G., Yasinskaya O.* Software system for the diagnosis of the spine diseases using case-based reasoning // Proceedings of the International Conference on Biomedical Engineering and Computational Technologies (SIBIRCON / SibMedInfo – 2015). 28–30 October 2015. P. 205–210.

7. *Пальчунов Д. Е., Яхъяева Г. Э., Ясинская О. В.* Программная система, осуществляющая case-based reasoning для диагностирования заболеваний позвоночника // Сибирский научный медицинский журнал. 2016. Т. 36, № 1. С. 97–104.

8. *Пальчунов Д. Е., Яхъяева Г. Э., Ясинская О. В.* Применение теоретико-модельных методов и онтологического моделирования для автоматизации диагностирования заболеваний // Вестник НГУ. Серия: Информационные технологии. 2015. Т. 13, вып. 3. С. 42–51.

9. *Пальчунов Д. Е., Яхъяева Г. Э.* Нечеткие алгебраические системы // Вестник НГУ. Серия: Математика, механика, информатика. 2010. Т. 10, № 3. С. 75–93.

10. *Пальчунов Д. Е., Яхъяева Г. Э.* Нечёткие логики и теория нечётких моделей // Алгебра и логика. 2015. Т. 54, № 1. С. 109–118.

11. *Schank R., Abelson R.* Scripts, Plans, Goals and Understanding. New Jersey, 1977. 248 p.

12. *Slade S., Henry P.* Case-Based Reasoning: A Research Paradigm // AI Magazine. 1991. Vol. 12, no. 1. P. 41–55. ISSN 0738-4602. DOI: http://dx.doi.org/10.1609/aimag.v12i1.883.

13. *Golding A. R., Rey M., Rosenbloom P. S.* Improving Rule-Based Systems through Case-Based Reasoning // Knowledge Creation Diffusion Utilization. 1991. April 1998.

14. *Kolodner J. L.* An introduction to case-based reasoning // Artificial Intelligence Review. 1992. Vol. 6, no. 1. P. 3–34. ISSN 0269-2821. DOI 10.1007/BF00155578.

15. *Watson I. A. N., Marir F.* Case-based reasoning: A review // The Knowledge Engineering Review. 1994. Vol. 9, no. 4. P. 327–354.

16. Aamodt A., Plaza E. CBR: foundational issues, methodological variations and system approaches // AI Communications. 1994. Vol. 7, no. 1. P. 39–59.

17. *Ram A., Santamaria J.* Continuous case-based reasoning // Artificial Intelligence. 1997. Vol. 90, no. 1–2. P. 25–77. ISSN 00043702. DOI 10.1016/S00043702(96)000379.

Материал поступил в редколлегию 10.08.2018

P. V. Myznikov

Novosibirsk State University
1 Pirogov St., Novosibirsk, 630090, Russian Federation

miznikov72@gmail.com

DEVELOPMENT OF THE CASE-BASED APPROACH OF WEB INTERFACES REVERSE REENGINEERING

The article is devoted to implementation of case-based reasoning in web development. Given personal practical experience in this field the author proposes to automate html/css markup building with aggregation of code samples from previous cases, which is very similar to case-based reasoning approach on a methodological level. Namely, the article describes a technique of construction of systems generating html code from a bitmap based on CBR. In particular, an original theory of building an image structure is presented in the article. Besides, the article considers modification of case-based reasoning approach that allows to get the result required. Also the results of practical experiments are described.

Keywords: case based reasoning, computer vision, optical character recognition, reverse engineering, web development.

References

1. DeepCoder: Learning to Write Programs. M. Balog [et al.]. 2016, Nov. arXiv: 1611.01989. URL: http://arxiv.org/abs/1611.01989.

2. TerpreT: A Probabilistic Programming Language for Program Induction. A. L. Gaunt [et al.]. 2016, Aug. arXiv: 1608.04428. URL: http://arxiv.org/abs/1608.04428.

3. Latent Predictor Networks for Code Generation. W. Ling [et al.]. 2016, Mar. arXiv: 1603.06744. URL: http://arxiv.org/abs/1603.06744.

4. Nguyen T. A., Csallner C. Reverse Engineering Mobile Application User Interfaces with REMAUI (T). *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Nov. 2015, p. 248–259. DOI 10.1109/ASE.2015.32.

5. Beltramelli T. pix2code: Generating Code from a Graphical User Interface Screenshot. 2017. arXiv: 1705.07962. URL: http ://arxiv.org/abs/1705.07962.

6. Palchunov D., Yakhyaeva G., Yasinskaya O. Software system for the diagnosis of the spine diseases using case-based reasoning. *Proceedings of the International Conference on Biomedical Engineering and Computational Technologies (SIBIRCON / SibMedInfo-2015)*. 28–30 October 2015, p. 205–210.

7. Palchunov D. E., Yakhyaeva G. E., Yasinskaya O. V. Software system for diagnosing spinal diseases using case-based reasoning. *Siberian Scientific Medical Journal*, 2016, vol. 36, issue 1, p. 97–104. (in Russ.)

8. Palchunov D. E., Yakhyaeva G. E., Yasinskaya O. V. Application of model-theoretic methods and ontological modeling to automate the diagnosis of diseases. *Vestnik NSU. Series: Information Technologies*, 2015, vol. 13, no. 3, p. 42–51. (in Russ.)

9. Palchunov D. E., Yakhyaeva G. E. Fuzzy algebraic systems. *Vestnik NSU. Series: Mathematics, Mechanics, Informatics*, 2010, vol. 10, no. 3, p. 75–93. (in Russ.)

10. Palchunov D. E., Yakhyaeva G. E. Fuzzy logics and fuzzy model theory. *Algebra and Logic*, 2015, vol. 54, no. 1, p. 74–80. (in Russ.)

11. Schank R., Abelson R. Scripts, Plans, Goals and Understanding. New Jersey, 1977. 248 p.

12. Slade S., Henry P. Case-Based Reasoning: A Research Paradigm. *AI Magazine*, 1991, vol. 12, no. 1, p. 41–55. ISSN 0738-4602. DOI: http://dx.doi.org/10.1609/aimag.v12i1.883.

13. Golding A. R., Rey M., Rosenbloom P. S. Improving Rule-Based Systems through Case-Based Reasoning. *Knowledge Creation Diffusion Utilization*, 1991, April 1998.

14. Kolodner J. L. An introduction to case-based reasoning. *Artificial Intelligence Review*, 1992, vol. 6, no. 1, p. 3–34. ISSN 0269-2821. DOI 10.1007/BF00155578.

15. Watson I. A. N., Marir F. Case-based reasoning: A review. *The Knowledge Engineering Review*, 1994, vol. 9, no. 4, p. 327–354.

16. Aamodt A., Plaza E. CBR: foundational issues, methodological variations and system approaches. *AI Communications*, 1994, vol. 7, no. 1, p. 39–59.

17. Ram A., Santamaria J. Continuous case-based reasoning. Artificial Intelligence, 1997, vol. 90, no. 1–2, p. 25–77. ISSN 00043702. DOI 10.1016/S00043702(96)000379.

Received 10.08.2018

For citation:

Myznikov P. V. Development of the Case-Based Approach of Web Interfaces Reverse Reengineering. *Vestnik NSU. Series*: *Information Technologies*, 2018, vol. 16, no. 4, p. 115–126. (in Russ.) DOI 10.25205/1818-7900-2018-16-4-115-126