ISSN 2079-3316 ПРОГРАММНЫЕ СИСТЕМЫ: ТЕОРИЯ И ПРИЛОЖЕНИЯ № 4(18), 2013, с.45—59

УДК 004.4'242

Е. В. Кочуров

Конструктивный синтез пользовательских интерфейсов Web-приложений

Аннотация. В статье рассматривается приложение метода конструктивного синтеза программ к созданию пользовательских интерфейсов Web-приложений. Для решения данной задачи построена специализированная конструктивная логика, которая позволяет синтезировать гарантированно правильные модели интерфейсов. Рассмотрена программная реализация и результаты практического использования модели.

Ключевые слова и фразы: пользовательский интерфейс, метод конструктивного синтеза программ, конструктивная логика, web-приложение, html-шаблон.

Введение

На сегодняшний день существует множество технологий и фреймворков для создания Web-приложений и, в частности, пользовательских интерфейсов, предназначенных для отображения в браузерах. И это не удивительно в связи с тем, что глобальные сети находят широчайшее применение во многих отраслях экономики, касается ли это потребительского, корпоративного или государственного сектора.

Процессы стандартизации языков HTML[1], CSS[2] и javascript[3] позволили достичь не только высокой степени кроссплатформенности пользовательских интерфейсов, но и достаточно хорошей степени кроссбраузерности, поэтому использование соответствующих стандартов при построении Web-приложений на сегодня стало доминирующим подходом.

Однако при разработке больших приложений одних только стандартов оказывается недостаточно: нужны шаблоны проектирования, библиотеки стандартных элементов управления, средства

© Е. В. Кочуров, 2013
© Институт программных систем им. А. К. Айламазяна РАН, 2013
© Программные системы: теория и приложения, 2013

поддержки логики представления (например, Presenter в модели MVP[4]) и многое другое. Соответствующие инструментальные средства все еще находятся в стадии становления, поэтому охватить весь стек необходимых технологий удается лишь в достаточно нишевых инструментах. В качестве хороших примеров из разряда свободно распространяемых продуктов можно привести MediaWiki[5], Drupal[6], WordPress[7]. Неизбежной расплатой для таких систем является привязка к серверным технологиям, что ограничивает возможности применения в ситуациях, когда серверное окружение для разработчика фиксировано.

Если рассматривать клиентские библиотеки, которые не зависят от серверных технологий, то четко прослеживается их специализация: на манипуляции с DOM-моделью (jQuery[8], Zepto.js[9]), стилевом оформлении страниц и элементах управления (Bootstrap[10], jQuery UI[11], w2ui[12]), построении каркасов приложений (AngularJS[13], Backbone.js[14], Knockout[15]).

Несмотря на богатый выбор среди существующих инструментальных средств, остается по меньшей мере одна задача, решение которой актуально при разработке больших систем: автоматическое порождение моделей интерфейсов по спецификации и контроль целостности данных моделей.

1. Постановка задачи

С практической точки зрения, механизм генерации web-форм, помимо функций работы с моделями интерфейсов, должен удовлетворять следующим требованиям:

- форма должна быть независима от источника данных;
- форма может быть как простым набором полей, так и содержать подразделы или таблицы;
- внешний вид форм должен быть настраиваемым;
- на странице может быть несколько форм, которые могут взаимодействовать;
- должно поддерживаться повторное использование форм в различных приложениях и разных частях одного приложения;

- должно быть возможным совместное использование решения с различными серверными и клиентскими технологиями.

1.1. Основные понятия

Элемент данных — фрагмент данных, хранимых сервером. Имеет древовидную структуру и может быть как атомарным значением (например, значением поля данных некоторой таблицы базы данных), так и составным — содержать вложенные структуры и массивы элементов данных. Для представления и транспорта элементов данных используется формат JSON. Из этого формата данные прозрачно преобразуются в javascript-объекты, поддерживаемые браузерами — основными клиентскими приложениями.

Элемент управления — фрагмент Web-документа, выполняющий одну или несколько функций:

- форматирование фрагмента Web-документа или всего документа целиком;
- навигация по компонентам Web-документа или между документами;
- организация (группировка, расположение, доступность) вложенных элементов управления;
- отображение элемента данных;
- редактирование элемента данных.

Таким образом, здесь понятие элемента управления трактуется значительно шире, чем это принято в спецификации HTML.

Шаблон — абстрактное описание элемента управления, на основе которого синтезируется конкретный экземпляр элемента управления с учетом контекста его использования. Шаблоны могут быть конструктивными — использовать внутри себя другие шаблоны.

Понятие шаблона также расширено по сравнению с устоявшимися в Web-разработке подходами: помимо стандартных компонент (HTML, CSS и javascript), шаблон имеет спецификацию, на основе которой конструктор форм включает шаблон в палитру доступных для настройки форм компонент, применяет и проверяет ограничения на использование шаблона.

Контекст — javascript-объект, описывающий место применения шаблона в момент синтеза конкретного элемента управления. Свойства контекста используются для адаптации внешнего вида и поведения элемента управления. Контекст включает в себя:

- Контекст данных (data) – элемент данных, для которого синтезируется элемент управления.
- Контекст шаблона (field) – который содержит конкретные значения всех описанных в спецификации свойств шаблона.
- Контекст формы (form) – форма, которая инициировала синтез элемента управления.

Форма — самостоятельный фрагмент пользовательского интерфейса с собственной логикой поведения, для отображения которого были использованы шаблоны.

Понятие формы сходно по назначению с определением из спецификации HTML, но гораздо свободнее в способах реализации и не ограничивается DOM-элементом типа FORM.

Класс объектов — платформозависимая сущность, которая увязывает источник данных с формой и описывает поведение данных и пользовательского интерфейса.

Конструктор форм — модуль, предоставляющий инструменты для конструирования шаблонов, классов объектов, визуального представления форм и логики поведения, а также контроля целостности получаемых моделей.

Генератор форм — набор клиентских javascript-библиотек, отвечающих за рендеринг форм на основе шаблонов и набора данных, которые следует отобразить, а также за исполнение клиентской логики поведения пользовательского интерфейса, описанной в конструкторе форм.

1.2. Модель вычислений

На уровне отдельно взятой формы основной принцип конструирования — структурно-иерархический. Это позволяет оформить отдельные повторяющиеся фрагменты интерфейсов как повторно используемые модули.

На уровне взаимодействия форм между собой структурные подходы уже не являются достаточно эффективными, т.к. лишь малое число форм находится в фиксированных отношениях вида «главный-подчиненный». Поэтому на данном уровне произведен переход к сетевой модели организации: формы являются независимыми действующими агентами, реагирующими на события друг друга.

1.3. Математическая модель

Сложность задачи контроля целостности модели формы усугубляется тем, что шаблоны, которые являются строительными блоками, сами представляют из себя трудноформализуемые конструкции: как только к синтезу модели подключается произвольный javascript (а это почти всегда допускается), надежда на разрешимость полезных свойств модели исчезает.

Чтобы отделить эти две проблемы (целостность шаблона и целостность модели формы) друг от друга, вводится специализированная конструктивная логика, предназначенная для синтеза моделей форм с учетом спецификации используемых шаблонов.

Введем язык конструктивной логики L = <P, C, Op, @>, где:

- P — множество пропозициональных переменных;
- C — множество констант {null, false};
- Op — множество связок {⇒, &};
- @ — отношение реализации.

Отношение реализации $f@F$ (читается как «f реализует F») устанавливает связь между выводимой формулой F и программой f, которая реализует объект, описываемый данной формулой. Реализацией формул данного языка являются функции-конструкторы на языке javascript. Подход к моделированию, при котором логические формулы реализуются сразу исполняемым программным кодом, минуя промежуточные математические объекты, развивается А.П. Бельтюковым[16]. Основная заимствованная идея, под влиянием которой разрабатывается данная математическая модель, заключается в том, чтобы перенести доказательство правильности из

логики непосредственно на класс программ, имеющих практическую применимость.

Реализация констант:

(1)
```
(function (c){
    if (c) throw "Ошибка!";
    return [];
}) @ null,
```

(2)
```
(function (){
    throw "Ошибка!";
}) @ false.
```

Реализациями атомарных формул являются стандартные шаблоны:

(3)
```
(function (c){ return [{
    type: "A",
    render: funcA,
    childs: c
}]; }) @ A,
```

где поле type хранит имя шаблона, поле render — ссылку на функцию рендеринга элемента управления по шаблону A, поле childs — массив вложенных шаблонов. Для ясности применения модели несущественные (с точки зрения конструирования) поля шаблона здесь опущены.

Правила вывода:

(4) $$\frac{A \Rightarrow B,\ B \Rightarrow C}{A \Rightarrow (B \Rightarrow C)},$$

(5) $$\frac{A \Rightarrow B,\ A \Rightarrow C}{A \Rightarrow (B \& C)},$$

(6) $$\frac{A \Rightarrow \text{null},\ B}{A \Rightarrow B},$$

(7) $$\frac{\text{null} \Rightarrow A}{\text{false}},$$

(8) $$\frac{(A \Rightarrow B) \Rightarrow C}{\text{false}},$$

(9) $$\frac{(A\&B) \Rightarrow C}{\text{false}}.$$

Если в некоторой теории выводима константа false, то такая теория является противоречивой.

Правила реализации:

Если $a@A$, $b@B$, то:

(10)
```
(function (c){
    if (c) throw "Ошибка!";
    return a(b());
}) @ A⇒B,
```

(11)
```
(function (c){
    if (c) throw "Ошибка!";
    return a().concat(b());
}) @ A&B.
```

Содержательно, данная логика позволяет конструировать деревья с типизированными узлами с учетом ограничений на типы дочерних узлов. Операция конъюнкции & является ассоциативной, но не коммутативной, что не позволяет считать данную логику фрагментом интуиционистской логики. Отказ от свойства коммутативности является сознательным шагом, т.к. порядок следования формул в конъюнкции важен для решаемой задачи.

Спецификация набора шаблонов формализуется логической теорией, которая строится в соответствии со следующей техникой:

(1) Каждому шаблону сопоставляется одноименная пропозициональная переменная и функция реализации.
(2) Если шаблон B может использоваться, как вложенный шаблон в A, то вводится аксиома $A \Rightarrow B$.
(3) Часто используемые в выводах подформулы рекомендуется вводить, как аксиомы.
(4) На практике бывает удобной запретительная стратегия спецификации шаблонов, когда по умолчанию разрешены любые вложения шаблонов друг в друга, кроме некоторых. В этом случае, чтобы избежать квадратичного роста числа аксиом с ростом числа шаблонов, используется следующий подход:
    a. Для каждого шаблона A вводится аксиома A.

b. Если в шаблон A могут вкладываться любые шаблоны, то вводится аксиома $A \Rightarrow \text{null}$, как сокращенная за счет правила вывода (6) запись множества импликаций.

Теорема о корректности.

(1) *Реализация любой выводимой формулы является синтаксически правильной функцией-конструктором на языке javascript.*
(2) *Реализацией вывода противоречия является функция, выполнение которой приводит к программному исключению.*
(3) *Реализация любой формулы, выводимой в непротиворечивой теории, является функцией, результат выполнения которой — это массив деревьев шаблонов, в котором каждое дерево удовлетворяет всем спецификациям.*

Доказательство.

Реализация константы null синтаксически правильна по определению (1) и возвращает пустой массив деревьев. Реализация атомарной формулы синтаксически правильна по определению (3) и возвращает массив с единственным элементом — деревом из единственной вершины. Реализация константы false синтаксически правильна и приводит к программному исключению по определению (2). Если реализации $a@A$, $b@B$ — синтаксически правильные javascript-функции, то реализации связок $\Rightarrow$ и $\&$ также являются синтаксически правильными функциями по определениям (10) и (11) соответственно.

Функция-конструктор получает непустой входной параметр тогда и только тогда, когда данная функция является реализацией посылки импликации. Следовательно, реализации формул $\text{null} \Rightarrow A$, $(A \Rightarrow B) \Rightarrow C$, $(A \& B) \Rightarrow C$ приводят к программным исключениям по определениям (1), (10) и (11) соответственно. Таким образом, правила вывода (7), (8) и (9) являются корректными. Из этого также следует, что посылкой импликации в непротиворечивой теории может быть только атомарная формула.

Пусть реализации $a@A$, $b@B$ — корректные функции-конструкторы. Тогда реализация формулы $A \Rightarrow B$ (с учетом того,

что в посылке стоит атомарная формула) дает корректную функцию-конструктор, которая, по определениям (3) и (10), присоединяет к a массив дочерних вершин b.

Пусть реализации $a@A$, $b@B$ — корректные функции-конструкторы. Тогда реализация формулы $A\&B$ (с учетом того, что любой конструктор возвращает массив) дает корректную функцию-конструктор, которая, по определению (11), выполняет конкатенацию массивов.

Пусть реализации $a@A$, $b@B$, $c@C$ — корректные функции-конструкторы, $A\Rightarrow B$ и $B\Rightarrow C$. Тогда, по определению (10), реализация формулы $A\Rightarrow(B\Rightarrow C)$ присоединяет к a в качестве дочерней вершины реализацию $B\Rightarrow C$, что соответствует спецификации $A\Rightarrow B$. Следовательно, правило вывода (4) корректно.

Пусть реализации $\mathrm{a}@A$, $\mathrm{b}@B$, $\mathrm{c}@C$ — корректные функции-конструкторы, $A\Rightarrow B$ и $A\Rightarrow C$. Тогда, по определениям (10) и (11), реализация формулы $A\Rightarrow(B\&C)$ присоединяет к a в качестве дочерней вершины конкатенацию массивов b и c, что соответствует принятым спецификациям. Следовательно, правило вывода (5) корректно.

Правило (6) корректно по принятым соглашениям использования константы null.

Конец доказательства.

1.4. Архитектура

В момент обращения к некоторому классу, серверная сторона механизма готовит необходимые данные, шаблоны, описания форм и javascript-код клиентской логики поведения формы. Подготовленная информация доставляется на клиентское устройство, где с помощью генератора форм преобразуется в интерфейс пользователя. Схема взаимодействия компонент механизма представлена на Рис. 1. Архитектура решения

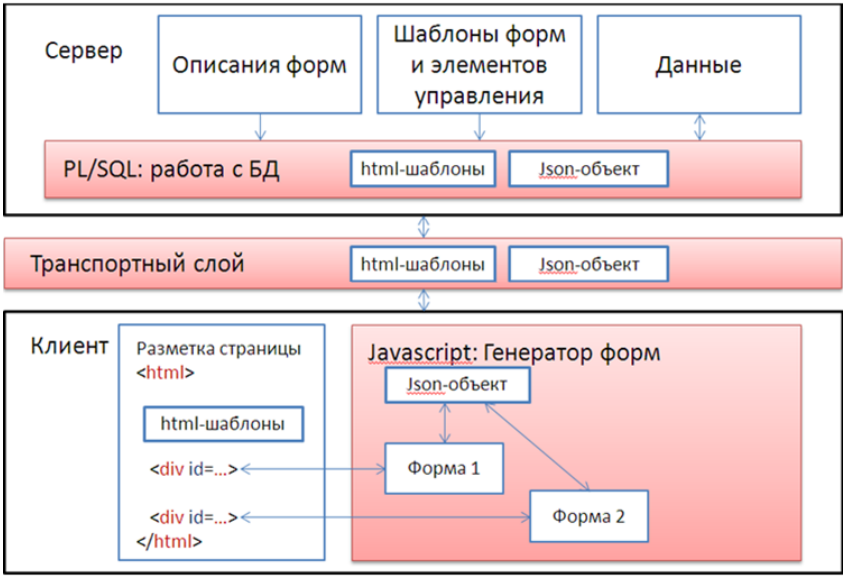

Рис. 1. Архитектура решения

2. Решение

Поскольку механизм изначально рассчитывался на разработку больших приложений, были разработаны инструментальные средства поддержки моделирования шаблонов, форм и классов.

Отдельное внимание можно уделить конструктору форм (Рис. 2. Конструктор форм), в котором была реализована построенная математическая модель. Конструктор состоит из двух частей:

- «Свойства формы» — описывает дерево шаблонов, используемых в данной форме. Для каждого шаблона настраиваются его свойства, состав которых определяется спецификацией шаблона;
- «Визуальное представление формы» — выполняет функцию предпросмотра настраиваемой формы, как она будет выглядеть для конечного пользователя.

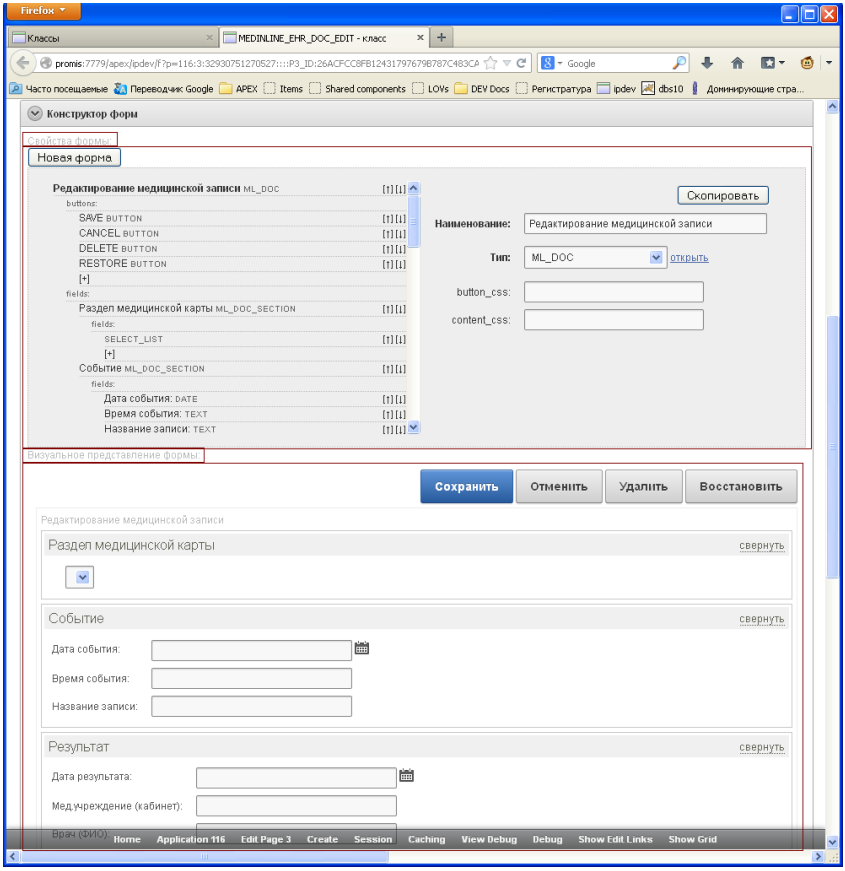

Рис. 2. Конструктор форм

В состав генератора форм входят следующие библиотеки:

- form-generator.js — платформо-независимое ядро механизма, решает следующие задачи:
  - компилирует шаблоны в исполняемый код (для ускорения рендеринга);
  - предоставляет фабрику форм;
  - предоставляет высокоуровневое API для манипулирования формами и данными из клиентской логики;

  - обеспечивает взаимодействие формы с DOM-моделью Web-документа.
- form-constructor.js — библиотека трансформации мета-моделей, используемая в процессе конструирования форм. Включает в себя:
  - конструкторы спецификаторов шаблонов;
  - синтез модели шаблона по его спецификации;
  - синтез модели элемента данных по модели шаблона;
  - прямой и обратный синтез примера элемента данных по его модели;
  - другие вспомогательные функции.
- events.js — форк фрагмента сторонней бибилиотеки backbone.js, который отвечает за функционирование событийных моделей приложения и форм. Включает в себя:
  - конструктор триггеров событий;
  - конструктор обработчиков событий.
- apex.js — реализует транспортный слой взаимодействия клиента с сервером:
  - загрузка форм и данных для них с сервера;
  - сохранение данных на сервер;
  - ajax-запросы к pl/sql-процедурам;
  - и другие APEX-специфичные функции.

Сторонние библиотеки:

- jQuery (обеспечивает слой манипуляций с DOM-моделью Web-документа[8]);
- underscore.js — коллекция утилит для поддержки функционального стиля программирования[17];
- beautify.js (предоставляет функции форматирования исходных кодов, используется в конструкторе форм[18]);
- backbone.js (используется фрагмент библиотеки в части событийной модели (см. events.js)[14]).

Все использованные сторонние библиотеки являются свободнораспространяемыми по лицензии MIT[19], допускающей в т.ч. коммерческое использование.

Внедрения и полученные результаты

Решение в данный момент имеет две инсталляции:

(1) Тестовое окружение, в котором ведется разработка механизма. Развернуто в облачном сервисе Google AppEngine.
(2) МИС Интерин PROMIS, в которой модуль web-регистратуры реализован на данном механизме. В качестве серверной стороны выступает Oracle Database.

Кроме того, в настоящее время ведется разработка медицинского портала, в котором данный механизм используется как надстройка над Oracle Application Express, способная заменять стандартный рендеринг страниц частично либо полностью.

Благодаря компиляции шаблонов время рендеринга исходного текста страниц в большинстве случаев является пренебрежимо малым по сравнению со временем, необходимым браузеру для разбора текста и отображения страницы, что положительно сказывается на отзывчивости пользовательского интерфейса. Огромное практическое значение имеет автоматизация синтеза гарантированно правильных моделей форм, т.к. отладка этого сорта ошибок является самым больным местом многих фреймворков.

Результаты использования показывают, что механизм конструктивного синтеза форм не только органично вписывается в уже существующие технологии построения Web-приложений, но и сам обладает достаточным потенциалом, чтобы в будущем стать ядром технологии разработки.

Список литературы

[1] HTML 4.01 Specification. URL: http://www.w3.org/TR/REC-html40/ (дата обращения: 30.09.2013).
[2] Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. URL: http://www.w3.org/TR/CSS2/ (дата обращения: 30.09.2013).
[3] ECMAScript Language Specification - ECMA-262 Edition 5.1. URL: http://www.ecma-international.org/ecma-262/5.1/ (дата обращения: 30.09.2013).

[4] Архитектура MVP. URL: http://www.gwtproject.org/articles/mvp-architecture.html (дата обращения: 30.09.2013).
[5] MediaWiki. URL: http://www.mediawiki.org/wiki/MediaWiki (дата обращения: 30.09.2013).
[6] Drupal - Open Source CMS | drupal.org. https://drupal.org/ (дата обращения: 30.09.2013).
[7] WordPress › Русский. URL: http://ru.wordpress.org/ (дата обращения: 30.09.2013).
[8] jQuery. URL: http://jquery.com/ (дата обращения: 30.09.2013).
[9] Zepto.js: the aerogel-weight jQuery-compatible JavaScript library. URL: http://zeptojs.com/ (дата обращения: 30.09.2013).
[10] Bootstrap. URL: http://getbootstrap.com/ (дата обращения: 30.09.2013).
[11] jQuery UI. URL: http://jqueryui.com/ (дата обращения: 30.09.2013).
[12] w2ui: Home | JavaScript UI. URL: http://w2ui.com/web/ (дата обращения: 30.09.2013).
[13] AngularJS — Superheroic JavaScript MVW Framework. URL: http://angularjs.org/ (дата обращения: 30.09.2013).
[14] Backbone.js. URL: http://backbonejs.org/ (дата обращения: 30.09.2013).
[15] Knockout : Home. URL: http://knockoutjs.com/ (дата обращения: 30.09.2013).
[16] Бельтюков А. П. *Дедуктивный синтез программ первого порядка* // Технологии информатизации профессиональной деятельности (в науке, образовании и промышленности) - ТИПД-2011 : тр. 3 Всерос. науч. конф. с междунар. участием, Ижевск, 8-12 нояб. 2011 г., Т.1, Удмуртский университет, 2011, 451-470.
[17] Underscore.js. URL: http://underscorejs.org/ (дата обращения: 30.09.2013).
[18] Online JavaScript beautifier. URL: http://jsbeautifier.org/ (дата обращения: 30.09.2013).
[19] The MIT License (MIT) | Open Source Initiative. URL: http://opensource.org/licenses/MIT (дата обращения: 30.09.2013).

Рекомендовал к публикации *д. ф.-м. н. С.В. Знаменский*

Об авторе:

Евгений Владимирович Кочуров
Аналитик в Интерин Технологии, аспирант Института программных систем им. А.К. Айламазяна РАН
e-mail: eugene_vk@mail.ru

Образец ссылки на публикацию:

Е. В. Кочуров. Конструктивный синтез пользовательских интерфейсов Web-приложений // Программные системы: теория и приложения: электрон. научн. журн. 2013. Т. 4, № 4(18), с. 45–59. URL: http://psta.psiras.ru/read/psta2013_4_45-59.pdf

E. V. Kochurov. Constructive synthesis of Web-based application user interfaces.

Abstract. The article discusses the application of the method of constructive program synthesis to create user interfaces for Web-based applications. Specialized constructive logic is introduced, which allows to synthesize models of guaranteed right user interfaces. We consider the software implementation and results of the practical application of the model. (*in Russian*)

Key Words and Phrases: user interface, method of constructive program synthesis, constructive logic, Web-based application, html-template.