

# Automatic Generation of Intelligent JavaScript Programs for Handling Input Forms in HTML Documents

Tetsuya Suzuki and Takehiro Tokuda  
Dept. of Comp. Science, Tokyo Inst. of Tech.  
Ohokayama, Meguro, Tokyo 152-8552, Japan

## Abstract

We present a system to generate JavaScript programs for user input validation and automatic computation on HTML documents from completely declarative descriptions. Its input is an HTML document with validation conditions and constraints, and its output is an HTML document with a JavaScript program for validation and constraint satisfaction. Our system makes it easy to develop and maintain client side programs for that purpose because there is no need to write any procedure. The generated programs not only provide interactive action and help to keep consistency among data in input forms on client side but also will reduce both network traffic and load on server side.

This technique can be applied to other electronic documents such as PDF documents, VoiceXML documents and Wireless Markup Language documents because client side programs are available in them.

**Keywords:** Web applications, constraint programming, software generator

## 1. Introduction

Nowadays, electronic documents such as Hyper Text Markup Language(HTML) [8] documents and Portable Document Format(PDF)[1] documents are much popular and are used as user interfaces of Web applications, business documents and so on. Some of them can contain programs for client side processing. For example, HTML, Wireless Markup Language(WML)[7], VoiceXML[2], PDF are such languages.

Client side program embedded in electronic documents not only provide interactive action and help to keep consistency among data in their input forms on client side but also will reduce both network traffic and load on server side. For example, they can check user input on HTML documents before submissions. As a result, they reduce times to submit invalid data and Web servers need not deal with the invalid data.

Our goal about this study is development of tools for several kinds of electronic document such that they generate client side programs for user input validation and automatic computation from declarative description. Such tools make it easy to develop and maintain client side pro-

gram embedded in electronic documents for that purpose because there is no need to write any procedure.

In this paper, we present a constraint compiler as such a tool for HTML documents. Fig. 1 shows its process. Its input is an HTML document with validation conditions and constraints among user input, and its output is an HTML document with a JavaScript[5] program for validation and constraint satisfaction.

The rest of this paper is organized as follows. In section 2, we explain constraints we deal with. In section 3, we present a simple example of both validation conditions and constraints. An overview of generated programs is given in section 4. Section 5 and section 6 are comparisons. Section 7 concludes this paper.

## 2. Constraints

A constraint is a relation among variables in general. In this paper, we use data-flow constraints with priority levels.

A data-flow constraint is a constraint which has procedures to realize itself, and the procedures are called as methods. For example, a constraint labeled " $x = y + z$ " will have three methods  $x \leftarrow y + z$ ,  $y \leftarrow x - z$  and  $z \leftarrow x - y$ . In the case of the method  $x \leftarrow y + z$ ,  $y$  and  $z$  are input variables and  $x$  is only output variable. If one of the methods is executed, the constraint is realized. As this example, we use data-flow constraints whose methods are single output methods.

Priorities of constraints are called strengths. We use three strengths **required**, **medium** and **weakest** from the strongest to the weakest. Constraints with **required** strength are constraints which must be satisfied. Constraints with other strengths are constraints which are expected to be satisfied.

We use the Blue constraint solver[3] for constraint satisfaction on client side. It is a simple constraint solver for data-flow constraints with priorities and satisfies constraints as follows. Blue selects one of the strongest constraint among constraints such that they have methods whose inputs have been computed in the constraint satisfaction process and whose outputs have not, and then it executes the method of the constraint to compute the output's value. Blue repeats the selection of constraints and the execution of methods until there is no constraint to be selected.

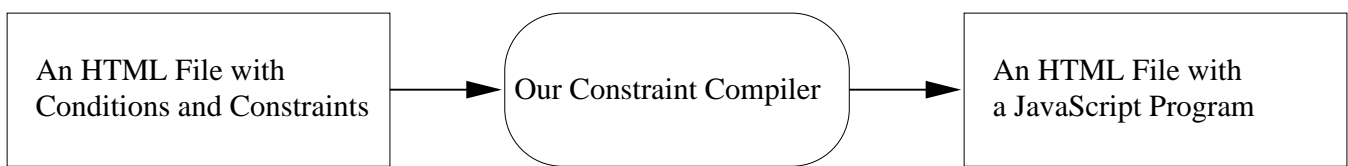


Figure 1. Compile process

### An HTML source document

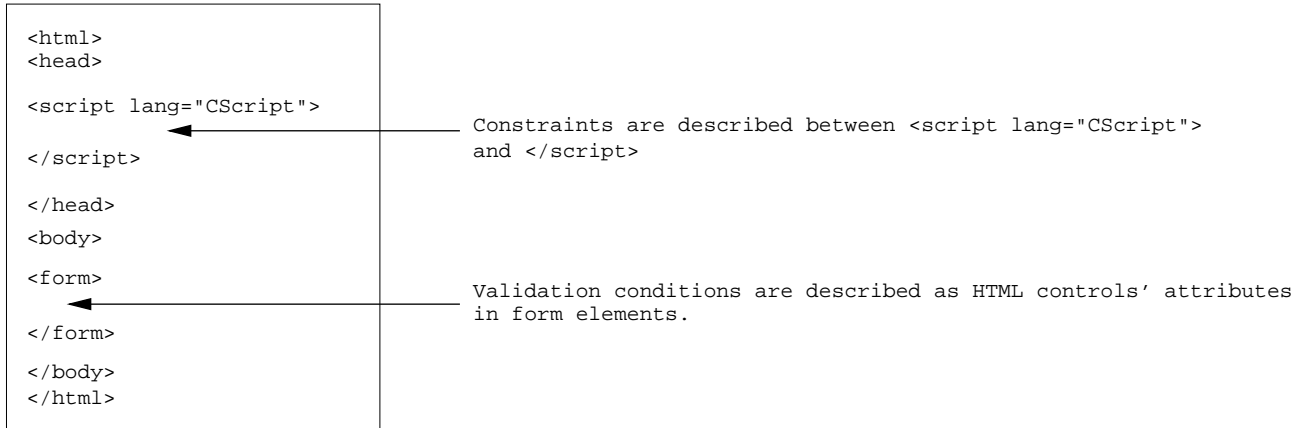


Figure 2. An overview of an HTML source document's structure

## 3. Description of Validation Conditions and Constraints

Fig.2 shows an overview of an HTML source document's structure. Constraints are described in the head element and validation conditions are described in the body element.

Fig.3 shows an HTML source document of a simple order form with validation conditions and constraints, and Fig.4 shows the Web page generated from the source. It has one select box to select a product and two text fields: one is for a number of the product and another is for the amount of costs. In the following, we explain the HTML source document.

Validation conditions of HTML controls' values are described as attributes' values of the HTML controls. For example, in the case of a text input field `<input type="text" domain="num?" >`, the attribute `domain` is not a standard attribute of HTML and specifies a condition of a string in the text field such that it must be a 0-length string or a number. Other available conditions are for lengths of strings, regular expressions, upper bounds and lower bounds when the values are viewed as numbers. It is also possible to combine these conditions.

All constraints are described in one `script` element whose `lang` attribute's value is "CScript" in the head element, and the description is organized as follows. At the beginning, there are statements which start

with a keyword `param`. Each of them specifies a binding between a constrained variable and an HTML control. A variable and an HTML control bound to each other always have same value. For example, the constrained variable `product` and the HTML control `form.product` are bound to each other in the source. The HTML control `form.product` is the `select` element whose name attribute's value is `product` in the `form` element whose name attribute's value is `form`. After the binding declarations, a constraint `ProductsCost` among products and their costs is defined by a statement which starts with a keyword `constraint`. Finally, two instances of constraints `ProductsCost(product, x)` and `amountOfCost := numOfProduct * x` are declared. The first is the user defined constraint, and the second is composed of two predefined constraints "==" and "\*". The "==" is a one-way equality constraint which propagates values from its right-hand side to its left-hand side. The "\*" is for multiplication. Other available constraints are "=", "+", "-", "/" and "@" for multi-way equality, addition, subtraction, division and concatenation of two strings respectively.

## 4. Generated Program

Fig.5 shows an overview of a generated HTML document's structure. Generated JavaScript functions are put into both the head element and the body element. The Blue constraint solver, arrays representing a given constraint prob-

```

<HTML>
  <head>
    <title>Order Form</title>
    <script lang="CScript">
      param product : form.product;
      param numOfProduct : form.numOfProduct;
      param amountOfCost : form.amountOfCost;

      constraint ProductsCost(String, Number) = {
        ("-", 0), ("P1", 100), ("P2", 200),
        ("P3", 300), ("P4", 400), ("P5", 500)
      };

      ProductsCost(product, x);
      amountOfCost := numOfProduct * x;
    </script>
  </head>
  <body>
    <h1>Order Form</h1>
    <form name="form" action="/cgi-bin/order.cgi">
      <table>
        <tr>
          <td>Product
            <select name="product">
              <option value="-">-
              <option value="P1">No.1
              <option value="P2">No.2
              <option value="P3">No.3
              <option value="P4">No.4
              <option value="P5">No.5
            </select>
          </td>
          <td>Number of Product
            <input type="text" name="numOfProduct" domain="num?">
          </td>
          <td>Amount of Cost
            <input type="text" name="amountOfCost" domain="num?">
          </td>
        </tr>
      </table>
      <input type="submit"><input type="reset">
    </form>
  </body>
</HTML>

```

Figure 3. An HTML document with validation conditions and constraints for a simple order form

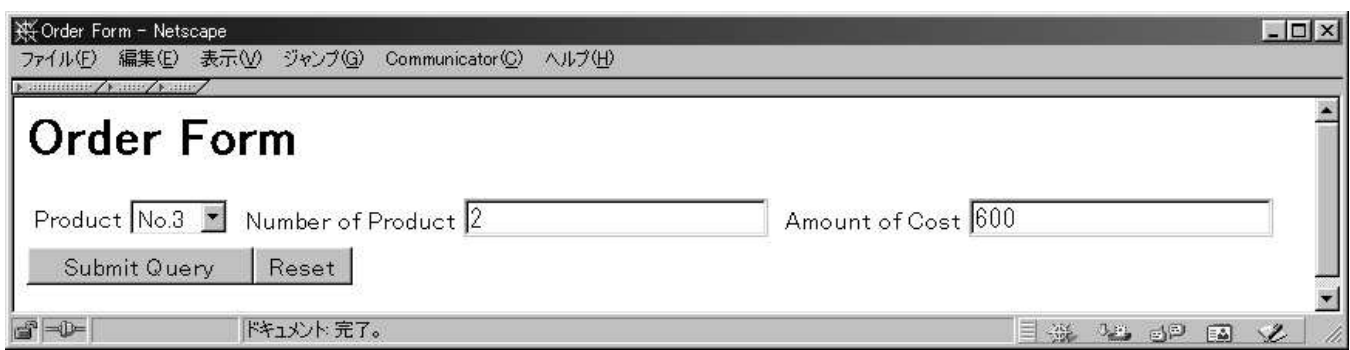


Figure 4. The generated Web page for the simple order form of Fig.3

lem, sub-functions and a main function are put into the head element. Initializer and handlers of events from Web browsers are put into the body element.

When a generated HTML document is loaded on a Web browser, the initializer in the document saves initial values of HTML controls with validation conditions or constraints. After that, whenever a value of the HTML controls is changed, an event handler is activated and the program behaves as follows.

1. It checks the validation condition. If the condition does not hold, it declares an error message, restore the saved value and go to the step 4.
2. If the validation condition holds, it tries to satisfy the constraints. If it can not satisfy the constraints, it declares an error message, restore the saved values and go to the step 4.
3. If the constraint satisfaction succeeded, it checks if all of the validation conditions hold in the solution. If a condition does not hold in the solution, it declares an error message and restore the saved values.
4. Finally, it saves the current values of HTML controls.

In the constraint satisfaction, constraints in HTML source documents are treated as **required** constraints while user input on Web browsers is set to variables by **medium** constraints. Current values are kept by **weakest** constraints.

For example, when a user selects a product or inputs a number of the product on the Web page of Fig.4, the amount of costs is automatically computed by the solver. If the user inputs a string which does not represent any number as a number of the product, the generated program will declare an error message and restore the previous values.

## 5. Implementation and an Experiment

We implemented our constraint compiler in Java[6] and used JavaCC[9] as a parser generator. The amount of lines of both the Java program and the grammar file for JavaCC is about 5,000 lines.

Table 1. A comparison of three HTML documents

	lines
The HTML source document	43
The generated HTML document	328
The handmade HTML document	60

We had an experiment using the simple order form. Table 1 compares the number of lines of three HTML documents: the HTML document of Fig.3, an HTML document generated from the HTML document of Fig.3 and an HTML document which has a handmade JavaScript program for the simple order form. In the following, we call them the HTML source document, the generated HTML document and the handmade HTML document respectively.

The HTML source document is shorter than the handmade HTML document. In addition, the description in the HTML source document is completely declarative. The description in the handmade HTML document is, however, complicated though the order form is simple. It is not only because it is procedural but also because the access methods to HTML controls' values depend on the kind of HTML controls. Fig.6 shows two methods to get values of HTML controls. One is for an input element and another is for a select element. The method for the input element is simple. The method for the select element is, however, complicated because one array, one index for the array and one property of the array's element are used to get one value.

The handmade HTML document is, however, short. It is about 18% of the generated HTML document in the number of lines. It is because the functions in the handmade HTML document are specific to the simple order form while the functions in the generated HTML document are generic.

## A generated HTML document

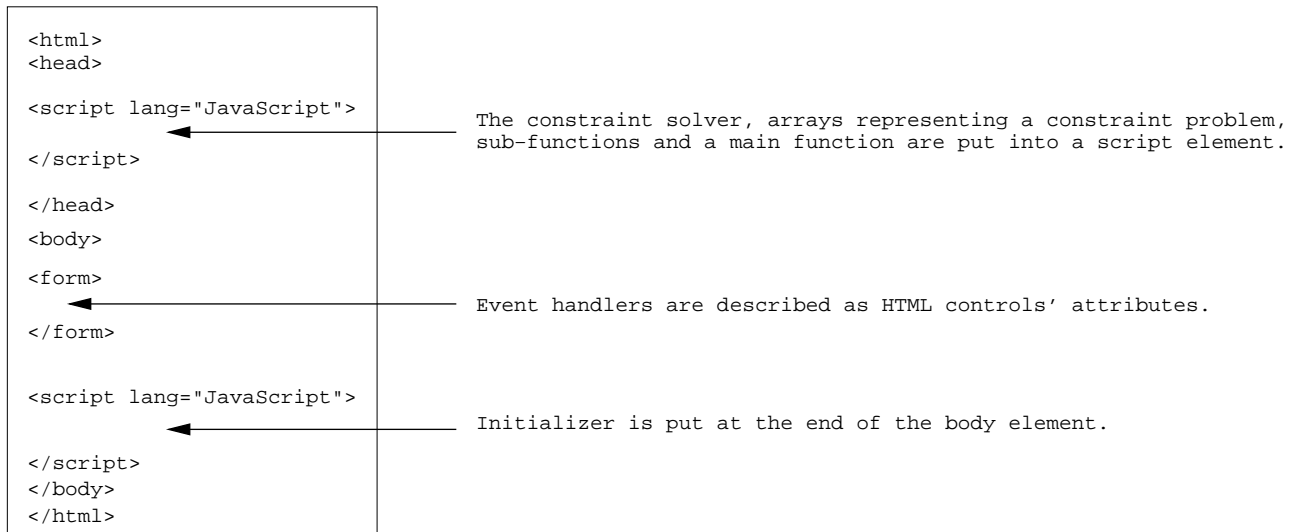


Figure 5. An overview of a generated HTML document's structure

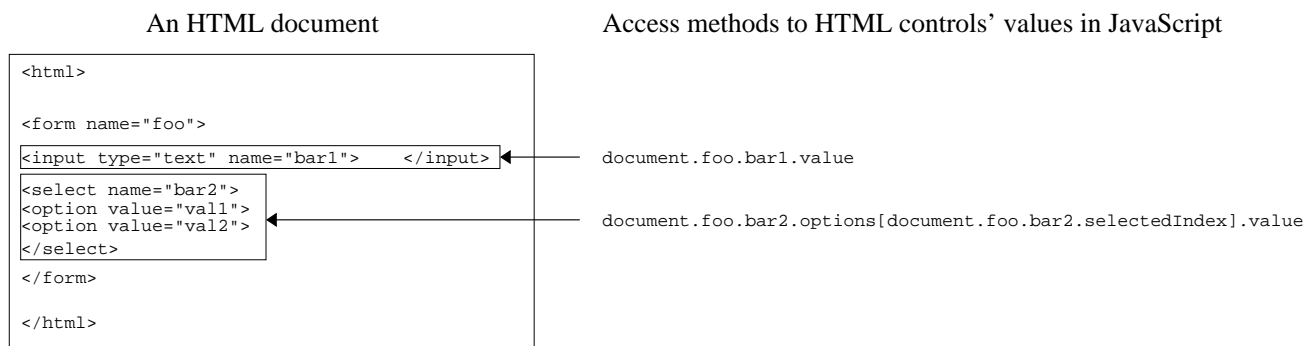


Figure 6. Access methods to HTML controls' values in JavaScript

## 6. Related Work

PowerForms[4] is a system similar to our constraint compiler. A point in common between PowerForms and our system is that they generate JavaScript programs from declarative descriptions for HTML documents' form fields. PowerForms's input is an HTML document with validation conditions and its output is an HTML document with a JavaScript program. The generated JavaScript programs check the validation conditions on Web browsers.

They, however, differ in the following points.

**Arithmetic computation** Our system deal with not only validation conditions but also arithmetic constraints, so that arithmetic computation on Web browsers is done by constraint satisfaction. PowerForms, however, does not support arithmetic interdependencies though it provides a way to make domain of variables dependent on other variables' values.

**Validation Condition description** PowerForms's users must name HTML controls which they want to attach validation conditions to. It is because validation conditions are written as one XML document and the XML document is put into the HTML document. In general, it is easy for programs to generate such XML documents but not for human. On the other hand, our system's users need not to name HTML controls because validation conditions are written as HTML controls' attributes. As a result, it is easier for human to describe validation conditions in our system than in PowerForms.

**Generated programs' behavior** In our system, HTML controls always have values which satisfy validation conditions and constraints if initial values are valid. It is because our system restores previous values if invalid values are given by users or computed by constraint satisfaction. On the other hand, in PowerForms, HTML controls with validation conditions can have invalid values before submissions though they must have valid values in submissions.

PowerForms provides a mechanism to let users know the status of input values incrementally. It attaches icons like traffic lights by text input fields and password input fields. Each of the icons changes its light according to the input value. It is green if the value is valid, yellow if the value is a proper prefix of valid values, and red otherwise. Our system, however, does not provide such a mechanism.

## 7. Conclusion

We presented a constraint compiler for HTML controls. Thanks to the compiler, once we describe validation conditions and constraints among HTML controls in an HTML source document, we can generate an HTML document with a Web client side program for the validation and

the constraint satisfaction. The generated program can work on many Web browsers because its target language is JavaScript.

Our future works is not only to improve our constraint compiler presented in this paper but also apply the technique used in the constraint compiler to other electronic documents such as PDF, WML, VoiceXML and so on.

## References

- [1] Adobe Systems Inc (Editor). *PDF Reference, Second Edition: Version 1.3*. Adobe Systems Incorporated, 2000.
- [2] Bob Edgar. *The VoiceXML Handbook: Understanding and Building the Phone-Enabled Web*. CMP Books, 2001.
- [3] Alan Borning. Programming language aspect of ThingLab: A constraint-oriented simulation laboratory. *ACM Transactions on Programming Languages and Systems*, 3(4):353–387, 1981.
- [4] C. Brabrand, A. Miller, M. Ricky, and M. Schwartzbach. Powerforms: Declarative client-side form field validation. *World Wide Web Journal*, 3(4), 2000.
- [5] Chuck Easttom. *Advanced Javascript*. Wordware Publishing, 2001.
- [6] James Gosling and Henry McGilton. The java language environment: A white paper. <http://java.sun.com/docs/white/langenv/>, Sun Microsystems Inc., May 1996.
- [7] Martin Frost. *Learning Wml & Wmlscript*. O'Reilly & Associates, 2000.
- [8] D. Raggett, A. L. Hors, and I. Jacobs. Html 4.01 specification. <http://www.w3.org/TR/html4/>, 1999.
- [9] WebGain, Inc. Javacc. [http://www.webgain.com/products/java\\_cc/](http://www.webgain.com/products/java_cc/).