社団法人　電子情報通信学会
THE INSTITUTE OF ELECTRONICS,
INFORMATION AND COMMUNICATION ENGINEERS

信学技報
TECHNICAL REPORT OF IEICE.
SS2002-22, KBSE2002-8(2002-10)

JavaScriptを含んだHTML文書に対するデータフロー解析を用いた構文検証手法の提案

鷲尾 和則† 松下 誠‡ 井上 克郎‡

†大阪大学大学院基礎工学研究科情報数理系専攻 〒560-8531 大阪府豊中市待兼山町 1-3
‡大阪大学大学院情報科学研究科コンピュータサイエンス専攻 〒560-8531 大阪府豊中市待兼山町 1-3
E-mail: †k-wasio@ics.es.osaka-u.ac.jp, ‡{matusita, inoue}@ist.osaka-u.ac.jp

あらまし JavaScriptを含んだHTML文書に対して，DTDに基づいたタグ付けが行われているかどうか，データフロー解析を用いてその構文を検証する手法を提案する．この手法では，JavaScript部分に対して，出力文に含まれる変数のデータフロー解析を行い，構文解析の結果と合わせて，出力される文字列を正規表現を用いてパターン化する．そしてそのパターンについて正規表現の包含関係を判定するなどして構文の検証を行う．

キーワード HTML, XHTML, XML, JavaScript, ECMAScript, 妥当性検証, データフロー解析

Syntax Verification Technique Using Data Flow Analysis for JavaScript Embedded HTML Documents

Kazunori WASHIO† Makoto MATSUSHITA‡ and Katsuro INOUE‡

† Division of Software Science Department of Informatics and Mathematical Science, Graduate School of Engineering Science, Osaka University 1-3 Machikaneyamacho, Toyonaka-shi, Osaka, 560-8531 Japan
‡ Graduate School of Information Science and Technology, Osaka University 1-3 Machikaneyamacho, Toyonaka-shi, Osaka, 560-8531 Japan
E-mail: †k-wasio@ics.es.osaka-u.ac.jp, ‡{matusita, inoue}@ist.osaka-u.ac.jp

Abstract This paper proposes that the technique of whether tag attachment based on DTD has been performed and verifying the syntax using data flow analysis is proposed to the HTML document containing Java Script. By this technique, data flow analysis of the variable contained in an output sentence is performed to a Java Script portion, the character sequence outputted is patternized together with the result of syntactic analysis using a regular expression, the inclusive relation of a regular expression is judged about the pattern, and syntax is verified.

Keyword HTML, XHTML, XML, JavaScript, ECMAScript, validation, data flow analysis

1. はじめに

現在，インターネットが広く普及し，膨大な量のホームページが作成されている．その多くはHyper-Text Markup Language (HTML) [1]によって記述されている．HTML文書は，タグという情報の区切りをいくつか並べて記述することで文書全体が構成されている．そのタグ付け規則は，HTML文書のDocument Type Definition (DTD: 文書型定義)で定義されており，すべてのHTML文書はDTDに従って記述することが求められる．もしDTDに違反したタグ付けが行われた場合，文書交換に支障を来す他，ウェブブラウザで正しく表示されないなどの問題が発生する．そのため，HTML構文がDTDにしたがって正しく記述されているかどうかを検証する必要がある．

一方，HTML文書にはJavaScript[2]などの言語で書かれたスクリプトを文書内に挿入することができる．スクリプトはクライアントまたはサーバーによって実行され，HTML文書の断片を出力する．主要なウェブブラウザがJavaScriptの表示をサポートしているため，JavaScriptは広く使われている．しかし，JavaScriptの構文はHTMLの文法とは独立しているため，しばしば誤った構文のHTML記述を出力するスクリプトを書いてしまうことがある．

現在，HTML文書の構文を検証する多くのツールが

存在する．しかしそれらのツールは JavaScript によって書かれたスクリプトの内容を無視するため，スクリプトの間違いを発見できない．そこで，JavaScript で書かれたスクリプトの内容を考慮して HTML 文書の構文の検証を行う必要がある．

その検証方法として我々が以前提案した手法[12]では，インタープリタを用いて JavaScript で書かれたスクリプトの出力文字列を得て，スクリプト以外の HTML 文書と合わせて構文の検証を行うものであった．しかし，JavaScript で書かれたスクリプトの実行結果(出力内容)は動的に変わりうるため，この方法では検証を実行するたびに結果が変わることもあった．ゆえに，完全に構文を検証するには，すべての実行結果を検証する必要があるが，すべての結果を検証することは現実的ではない．

そこで，今回提案する構文の検証方法は，JavaScript が出力する内容を正規表現を用いてパターン化し，そのパターンを検証することで JavaScript によって動的に生成される HTML 文書の構文の検証を行う．

本手法により，すべての実行結果を検証せずに，動的に生成される HTML 文書の構文の検証が行える．

なお本研究では，HTML 文書として構文が厳密に定められている eXtensible Markup Language (XML) [3]で記述した eXtensible Hyper-Text Markup Language (XHTML) [4]文書を対象とする．また，JavaScript としては European Computer Manufacturer Association (ECMA) によって標準化されているスクリプト記述言語 ECMAScript[6]を対象とする．

2. XHTML の妥当性検証

本節では，XHTML 文書の構文の正しさを検証することの必要性と，検証に関する研究について述べる．

なお，これ以降，DTD のタグ付け規則にしたがって，構文が正しく記述されているかどうかを検証することを妥当性検証と呼ぶ．

2.1. 妥当性検証の難しさ

XHTML 文書はタグ (tag) という情報の区切りを組み合わせることで構成される．タグには開始タグと終了タグがあり，それらで囲まれた部分が一区切りの情報である．この情報のことを要素 (element) という．

ところで，一般的な XHTML 文書の妥当性検証手法は，ECMAScript の内容を単なるテキスト要素とみなし，内容は全く無視して検証する．よって ECMAScript 以外の部分が正しい構文で記述されていて，ECMAScript に間違った構文が記述されていても，その間違いを指摘することはない．つまり一般的な妥当性検証プログラムは，スクリプトなどが生成する XHTML の内容に対して妥当性を検証できない．

2.2. 関連研究

動的に生成される HTML 文書や XML 文書について，その妥当性を検証する研究がいくつかなされている．

Claus Brabrand らは<bigwig> [7] [8] という HTML 文書や XML 文書のためのスクリプト記述言語を対象として，スクリプトが動的に生成する HTML (XHTML) 文書の妥当性検証を行った．

Martin Kempa, Volker Linnemann は DOM を拡張し，DOM 自身に妥当性を保証する仕組みを持たせた V-DOM[9]と，V-DOM 用の XML 処理言語である P-XML[9]を開発した．

Erik Mejier, Mark Shields は XML 処理用の関数型言語 XMλ[10]を開発した．また，細谷晴夫，Benjamin Pierce は XML 処理のための型付き関数型言語 XDuce [11]を開発し，型チェック機能を導入しプログラムが生成する XML 文書の妥当性を保証するようにした．

いずれの研究も，XML 文書や HTML 文書を生成するために新しい言語を作り，その言語に妥当性を保証するような仕組みを構築している．しかし，プログラム言語として独自の言語を対象としており，実用的とは言えない．そこで，既存の言語を対象として妥当性検証を行うことが必要である．

3. 提案手法

提案する妥当性検証手法は，XHTML 文書を入力とし，検証結果を出力とする．具体的には XHTML 文書中にある ECMAScript が出力する文字列をパターンで表し，そのパターンを検証する．この手法により，動的に変わりうる出力文字列をパターン化して検証することで，すべての実行結果を検証せずに済む．しかし，動的に変数の値が決まるものは，静的には検証できないため，その値を検証することはできない．よって，本手法ではそれらの値については，不定値として検証する．

本手法の概略を図 1 に示す．この手法は大きく分けて次の 3 つの段階からなる．

フェーズ 1．XHTML 文書解析

まずは，対象となる XHTML 文書について構文解析を行い，木構造のデータに変換して，XHTML 記述と ECMAScript 部分を分離する．

フェーズ 2．スクリプト解析

次に，ECMAScript で書かれたスクリプトの解析を行う．これは構文解析及び出力文に含まれる変数のデータフロー解析が含まれる．それらの結果を元に，繰り返し(*)や選択(|)などの正規表現を用いて出力文字列をパターンとして表現する．

フェーズ３．パターンの妥当性検証

最後に，フェーズ２で得られたパターンを DTD に基づいて妥当性の検証を行う．

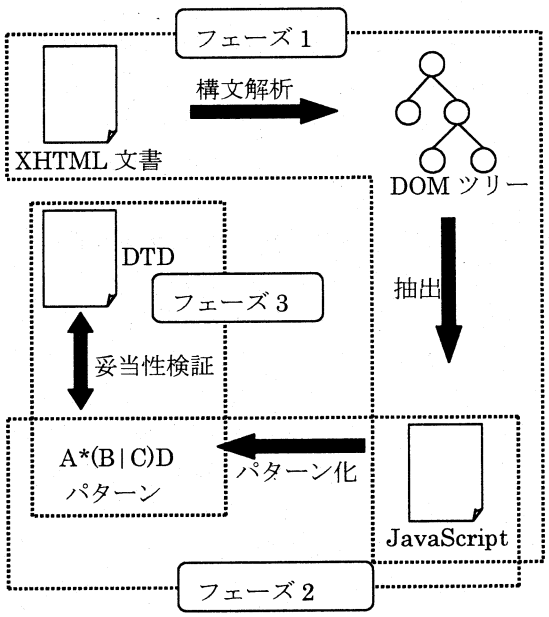


図 1 検証手法の概略図

以下，３つのフェーズについて順に説明する．

3.1. フェーズ 1: XHTML 文書解析

フェーズ１では，XHTML 文書の構文解析を行う．解析後，XHTML 記述と ECMAScript を分ける．ここで得られる ECMAScript は次のフェーズ２へ渡す．

最初に XML パーサで XHTML 文書を構文解析すると，Document Object Model (DOM) [5]ツリーと呼ばれる木構造のデータに変換できる．この DOM ツリーにより，要素の親子関係や，内容などの情報を得ることができる．また，このとき妥当性検証も同時に行う．この場合の妥当性検証は，スクリプトの内容を単なるテキストとみなして行う．これは一般的な検証プログラムと同等の処理である．

3.2. フェーズ 2: スクリプト解析

フェーズ２では，スクリプトの解析を行う．本フェーズは，1)ECMAScript で書かれたスクリプトの構文解析を行い，2)出力文に含まれる変数に対してデータフロー解析を行う．3)先ほどのデータフロー解析から得た変数の値をもとに，出力される文字列を，構文解析で得られた制御構造により正規表現で表すという３つの手順で行われる．この正規表現は次のフェーズ３で用いられる．

3.2.1. スクリプト構文解析

ECMAScript の構文定義をもとにスクリプトの構文解析を行い，スクリプトの抽象構文木を作成する．そして，if 文や while 文などの制御構造を判別する．これは後述のパターン化において必要となる．

3.2.2. データフロー解析

データフロー解析は，参照された変数がプログラム中のどの位置で定義されているのかを解析するものである．この解析を行うため，まずスクリプトに対して構文解析と意味解析を行う．そして，スクリプトを順に追って，変数が定義されれば変数表にその変数名や位置などの情報を追加する．変数が参照されれば変数表にその変数があるか確かめる（図 2）．これにより変数および定義されている位置と参照されている位置について記したリストを得ることができる．この際，変数のスコープなどの問題に注意する必要がある．

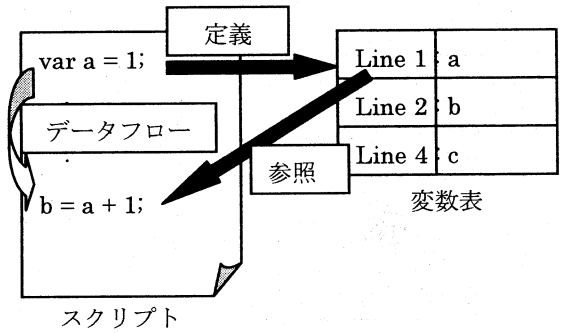


図２　データフロー解析

3.2.3. 出力文字列のパターン化

スクリプト解析の最終段階は，出力文字列を正規表現で表し，パターン化することである．ここでパターンについて説明する．例えば if 文の条件分岐により，一方の実行経路では"A"という文字列が出力され，他方の実行経路では"B"という文字列が出力されるとする．その両方の出力をまとめて記述するために，正規表現を用いる．この場合，選択記号の|を用いて，"A|B"というパターンとなる．同様に while 文や for 文の繰り返しについては，繰り返し記号の*を用いて，例えば"C*"というパターンで表す．

このパターン化は，出力文に含まれる変数についてデータフロー解析を行い，変数の処理を解析することで行える．ところで出力文の引数としては，固定文字列，文字列型変数，整数型変数，配列(全体)やオブジェクトを取ることができるが，配列，オブジェクトはそれを引数に取った場合，ブラウザやインタープリタが行う処理が不定なのでここでは省く．つまり，出力文に含まれる文字列型変数と整数型変数に対してデー

タフロー解析を行い，変数処理の流れ，つまり一連の変数の定義・参照関係の情報を得る．

そして，データフロー解析後，変数の処理の中で，その値が静的に定まればその値を変数の値として用いる．しかし，静的に定まらない，すなわち動的に定まる場合，その変数の値は不定値とし，その変数の型で表すことにする．また，変数の処理の中でクラスライブラリのメソッドを用いた処理があった場合，その変数の値は同じく不定値とする．

こうして得られた出力文に含まれる変数の値について，抽象構文木の情報からそれぞれの出力文が if 文や while 文などの制御構造下にある場合，正規表現で用いられる選択記号(|)や繰り返し記号(*)を付加する．選択記号(|)を付加する制御構造は，if 文，switch 文である（図３）．繰り返し記号(*)を付加する制御構造は，while 文，for 文である（図４）．また，それぞれの出力文字列の間には連結記号(,)を挿入する（図５）．

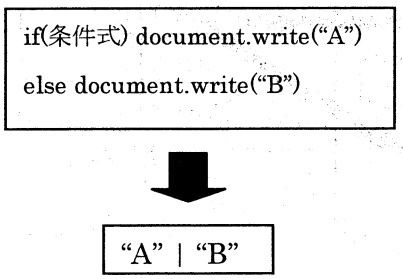


図３ 選択記号の付加

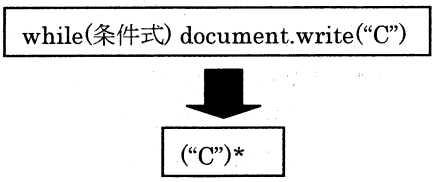


図４ 繰り返し記号の付加

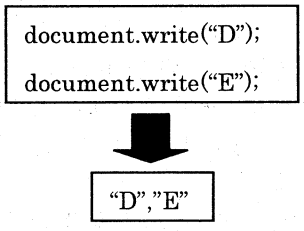


図５ 連結記号の付加

以上をまとめると，以下のアルゴリズムとなる．

アルゴリズム　出力文字列のパターン化
入力：ECMAScript によって書かれたスクリプト
出力：出力文字列のパターン
Step 1. スクリプトの構文解析および意味解析を行う．
Step 2. スクリプトを順に追いながら
Step 3. 出力文が含まれるブロックについて，そこに含まれるすべての出力文に対して Step4 を繰り返す．
Step 4-1. 出力文に含まれる変数について，データフロー解析を行う．変数処理を順に追い，値を求める．動的に決まるものは不定値とする．
Step 4-2. 出力文に含まれる固定値はその値を得る．
Step 5. Step4 で得られた値に対し，出力文が含まれるブロックの制御構造に対応して，|や*を付加する．
Step 6. すべてのブロックで Step3～5 を繰り返す．

3.3. フェーズ 3. パターンの妥当性検証

フェーズ３ではフェーズ２で得られたパターンの妥当性検証を行う．しかしフェーズ２で得られた，スクリプトの出力文字列を正規表現で表したパターンのままでは妥当性検証を行えない．そこで本フェーズでは，1)正規表現で表された文字列から，開始タグと終了タグのマッチングを行い，要素を特定し，2)要素が特定できた場合，フェーズ２で得られたパターンを，要素の並びを正規表現で表したパターンへと変換する．3)この要素の並びのパターンについて，DTD の親子関係を正規表現で表した定義と，包含関係を判定することで妥当性検証を行うという３つの手順を経る．以下，３つの手順について説明する．

3.3.1. 開始タグと終了タグの対応付け

正規表現で表した出力文字列に対して，要素を特定するために，まずは開始タグと終了タグの対応付けを行う必要がある．これは，最も内側の括弧内の文字列から，それぞれのタグの対応付けを行い，要素を特定する．また，特定したそれぞれの要素の親子関係も判別する．以下にそのアルゴリズムを示す．

アルゴリズム　パターン文字列の解析
入力：正規表現で表されたパターン文字列
出力：正規表現で表された要素の列
Step 1. 最も内側の括弧から
Step 2. 開始タグと終了タグの対応付けを行う．対応付けの行い方は後述する．対応付けできた要素を得る．同じ括弧内で対応付けが行えなくなるまで繰り返す．
Step 3. 括弧に選択記号(|)や繰り返し記号(*)があれば，Step 2 で得られた要素にその記号を付加する．
Step 4. 外側の括弧に移行し，Step 2,3 を行う．ここで得られた要素の子要素は，Step 3 で得られた要素の列である．括弧が無くなるまで繰り返す．

ここで対応付けを行うに当たって，開始タグと終了タグの間に括弧や|，*が無ければ，そのまま対応付けを取ることができる．図６の場合，要素 A の開始タグ

と終了タグの対応付けが取れ，要素Aを得る．

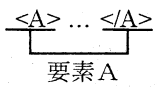


図6　要素判別（通常）

開始タグや終了タグに|が付いている場合，すべての組み合わせで対応付けを行う．図7の場合，要素AとBの開始タグに対して，要素BとCの終了タグがある．その組み合わせで，要素Bでの対応付けが取れる．

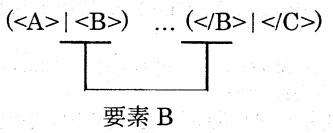
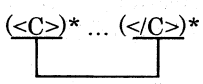


図7　要素判別（選択）

タグに*が付いている場合，開始タグ・終了タグともに付いており，かつ子要素にその要素自身が取れる(親要素と子要素が同一)なら，その要素のネストと判別することができる．どちらか一方のタグにのみ*が付いている場合は，構文エラーとみなす．図8の場合，要素Cの開始タグ・終了タグに繰り返し記号が付いているので，要素Cのネストである．



(C自身がCの子要素となりうる時)

図8　要素判別（繰り返し）

このように開始タグと終了タグの対応付けを行い，要素レベルの正規表現にする．

ところで，タグ記号そのものに*や|の記号が付加されている場合は，対応付けが困難になる．そのような場合は，タグ記号を制御構造の下にある複数の出力文にまたがって記述した場合であり，多くのユーザーはタグ記号を一つ出力文に書くのが一般的である．それゆえ，ここではその場合を除外する．

3.3.2. 要素の列の妥当性検証

正規表現で表された要素の列に対して，DTDの親子関係の定義との検証を行う．DTDで定義されている親子関係も正規表現で表されているので，その検証については，正規表現の包含関係の判定問題に帰着できる．

簡単化のため，包含関係の判定の際は要素列の正規表現において，

$$a^* = \varepsilon \mid a \mid aa$$

と，*を|を用いて置き換える．この置き換えを行っても，妥当性検証に支障はない．なぜならば，DTDでの繰り返しの正規表現は，*(0回以上)，+(1回以上)，?(0回あるいは1回)の3つの場合のみであるからである．このため，連続する要素数が0個，1個，2個の3つの場合について，どの正規表現にマッチするかを決定することができる．この置き換えでは，要素列の正規表現が選択記号のみ含まれるとなる．選択記号のある正規表現をすべての場合について文字列に展開し，文字列と正規表現のマッチングを行いやすくできる．展開してできたすべての文字列が，DTDの正規表現にマッチすれば，その構文が正しいと示すことができる．以上をまとめると，以下のアルゴリズムとなる．

アルゴリズム　パターン化した要素列の妥当性検証
入力：正規表現で表された要素列
出力：検証結果

Step 1. 要素列において，繰り返し記号(*)を選択記号(|)に置き換える．

Step 2. ルート要素から

Step 3. その要素の子要素の定義をDTDから得る．子要素のパターンについて，選択記号(|)があればすべて展開する．展開したすべての文字列に対して，DTDの正規表現とパターンマッチを行う．1つでもマッチしなければ，妥当性は保証されない．

Step 4. 次の子要素について，Step 3を行う．すべての要素について繰り返す．

Step 5. すべての要素で妥当性が保証されれば，XHTML文書全体の妥当性が保証できる．

4. 検証例

ECMAScriptのパターン表現の妥当性検証について例を示す．図9に検証用のサンプルスクリプトを示す．

```
var a = "good morning"
var b = "good afternoon"
Date day = new Date();
document.write("<UL>")
if(day.getHour() < 12){
   document.write("<LI>");
   document.write(a);
   document.write("</LI>");
}else{
   document.write(b);
}
document.write("</UL>");
```

図 9 ECMAScript サンプルスクリプト

サンプルスクリプトの出力文字列をパターン化すると以下のようになる.

"<UL>", (("<LI>" , "good morning" , "</LI>") | "good afternoon") , "</UL>"

このパターンについて，開始タグと終了タグの対応付けを取ると，図 10 のような親子関係ができる.

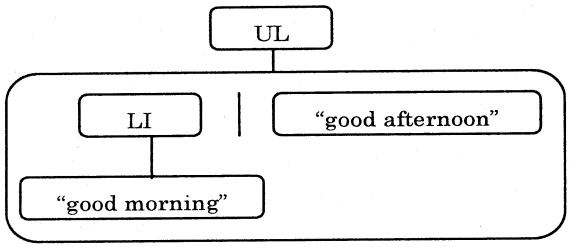


図 10 要素の親子関係

この親子関係をもとに XHTML の DTD と比較して妥当性検証を行う.

1. UL 要素を検証する. 子要素(LIIテキスト)に対し，その定義は LI+(1 回以上の LI)である. LI は違反しないが，テキストは違反する.
2. LI 要素を検証する. 子要素テキストに対し，その定義は#PCDATA(テキスト)である. これは違反していない.

したがって，このスクリプトは UL 要素の子要素としてテキストが出力される場合，すなわち else 節が実行されると DTD 違反となることがわかり，スクリプトの誤りを発見できる. 例えば，午前中にこのスクリプトを実行していたなら発見できなかったかもしれない誤りを，本手法によって発見することができる.

さらに，スクリプトのルート要素（複数存在することもある）について，他の兄弟要素との妥当性検証を行う. それにより，文書全体の妥当性を検証できる.

5. まとめと今後の課題

ECMAScript で書かれたスクリプトを含んだ XHTML 文書について，スクリプトに対してデータフロー解析を用いて出力文字列を正規表現でパターン化し，そのパターンを検証することで，その構文の妥当性を検証するための手法を提案した.

この検証方法により，ECMAScript のすべての実行結果について妥当性検証しなくとも，その構文を検証することができるようになる.

また，近年頻繁に利用されるようになった XML 文書についても，XML 処理プログラムが出力する文書に対して，同様の方法でその妥当性を検証することが可能であると考えられる.

また，今後の課題として，本手法を実際のシステムに実装し，検証精度などについて評価を行う必要がある.

文　献

[1] W3 Consortium, HTML 4.01 Specification, http://www.w3.org/TR/1999/REC-html401-19991224, 24 December 1999

[2] Netscape Communications Corporation, JavaScript 1.1 Language Specification, http://www.netscape.com/eng/javascript/index.html, 1997.

[3] W3 Consortium, Extensible Markup Language (XML) 1.0 (Second Edition), http://www.w3.org/TR/1998/REC-xml-20001006/, 6 October 2000.

[4] W3 Consortium, XHTML 1.0: The Extensible HyperText Markup Language (Second Edition), A Reformulation of HTML 4 in XML 1.0, http://www.w3.org/TR/2002/REC-xhtml1-20020801, 1 August 2002.

[5] W3 Consortium, Document Object Model (DOM) Level 1 Specification, Version 1.0, http://www.w3.org/TR/1998/REC-DOM-Level-1-19981001, 1 October 1998.

[6] ECMA – Standardizing Information and Communication Systems, Standard ECMA-262, ECMAScript Language Specification, 3rd edition, http://www.ecma.ch/ecma1/STAND/ECMA-262.HTM, December 1999.

[7] Claus Brabrand, Anders Moeller, and Michael I. Schwartzbach. The <bigwig> project. Submitted for publication. Available form http://www.brics.dk/bigwig .

[8] Claus Brabrand, Anders Moeller, and Michael I. Schwartzbach. Static Validation of dynamically generated HTML, SIGPLAN Notices, Supplement issue, pp.38-45, 2001.

[9] Martin Kempa and Volker Linnemann, V-DOM and P-XML – towards a valid programming of XML-based applications, Information and Software Technology, vol.44, Issue 4, pp.229-236, 31 March 2002.

[10] Erik Meijer and Mark Shields, XMλ:A functional language for constructing and manipulating XML documents, Draft, 1999.

[11] Haruo Hosoya and Benjamin Pierce, "XDuce: A Typed XML Processing Language", World Wide Web and Databases. Third International Workshop WebDB 2000, Lecture Notes in Computer Science vol.1997, pp.226-244, Dallas, Texas, May 2000.

[12] 鷲尾和則, 松下誠, 井上克郎, "JavaScript を含む HTML 文書の妥当性検証手法の提案", 2002 年電子情報通信学会総合大会講演論文集, D-3-5, pp.31, March 2002.