Journal of Information Processing Vol.20 No.3 622–632 (July 2012)

[DOI: 10.2197/ipsjjip.20.622]

Regular Paper

Automatic Generation of XML Files and Their Database Registration from Tabular Form Specifications

Yasunori Shiono[1,a)] Tomokazu Arita[2] Youzou Miyadera[3] Kimio Sugita[4]
Takeo Yaku[5] Kensei Tsuchida[1]

Received: September 14, 2011, Accepted: February 3, 2012

Abstract: Various forms of tables have been used as tools for visualizing and arranging information in many fields. In addition, XML is widely used as a language for exchanging data. We have studied how documents are formally processed with software development tools. In this paper, we propose a system to create and manage tabular specifications based on an attribute graph grammar. A tabular form specification is represented by a marked graph, and its syntax is defined by an attribute NCE graph grammar. We add a new attribute that contains XML source codes of the tabular form specifications. The XML source codes are generated by evaluating the attribute and are automatically registered to the database. The specifications are then retrieved from the database. Our system can perform a characteristic retrieval for software specifications. The results may lead to a considerable improvement in the efficiency of human labor due to the use of a unified formal methodology based on graph theory and advanced retrieval.

Keywords: hiform specifications, XML database, attribute graph grammar, parser, software information

1. Introduction

In recent years, the importance of user interfaces has been steadily increasing with the development of fundamental technology, widespread use of IT devices, and user needs. Under these circumstances, tables for displaying information in a user interface and in documents play an important role. Various tables have been used as tools for visualizing and arranging information in many fields. Such tabular form documents are created, referred to, and managed for their intended use. Many table processing systems have been developed, and tables are often used in computer interfaces and documents [1], [2], [3]. Tabular form specifications are also used for various developments. Conventionally, input, creation, and checking of tabular form specifications are processed manually. Document processing is dependent on human labor, though, and the percentage of processing automatically is comparatively low; therefore, machine-based document processing, such as automatic drawing and editing of tabular form specifications, is considered an important issue in a software development tool. The graph syntax theory aiming for global diagrammatic structures has recently been developed, and therefore, the possibility of automatic diagrammatic processing has arisen.

Software documentation often involves tabular forms, such as tabular form specifications, and diagrams, such as program flowcharts. Furthermore, the tabular forms may be classified into two groups, as follows.

(1) nested-structured forms in which items are linked hierarchically to one another.
(2) tessellation-structured forms such as symbol tables and spreadsheets.

This paper deals with (1) (that is, the nested-structured tabular forms) together with their mechanical manipulating problems.

In table processing systems, it is necessary to explicitly define both the syntax and drawing conditions. Attribute graph grammars formulate syntactic structures [4] and universally formulate visual structures among items in form using syntax with attribute rewriting rules.

Several models and properties of graph grammars have been investigated by Franck, Della Vigna, and Rozenberg [5], [6], [7], [8], [9]. Franck [5] introduced precedence graph grammars and applied them to nested program tabular forms called PLAN2D. In addition, graph grammars with Neighbourhood Controlled Embedding (NCE graph grammars) [7] have been considered as reasonable models of design and analysis. Adachi et al. have formulated hierarchical program diagrams by applying an attribute graph grammar [10], [11]. In accordance with the development of the graph grammar theory, syntactic graph manipulating systems have also been developed such as DIAGEN [8]. Another system introduced by Nagl et al. in the IPSEN project [8].

On the other hand, XML is widely recognized as one of the most influential standards concerning data exchange on the Web.

1 Toyo University, Kawagoe, Saitama 350–8585, Japan
2 J.F. Oberlin University, Machida, Tokyo 194–0294, Japan
3 Tokyo Gakugei University, Koganei, Tokyo 184–8501, Japan
4 Tokai University, Hiratsuka, Kanagawa 259–1292, Japan
5 Nihon University, Setagaya, Tokyo 156–8550, Japan
a) shiono@toyo.jp

The results of this paper partly appeared in "Syntactic Processing of Diagrams by Graph Grammars," by Arita, T., Tomiyama, K., Yaku, T., Miyadera, Y., Sugita, K. and Tsuchida, K. which appeared in *Proc. IFIP WCC ICS 2000*, pp.145–151 (2000), and "An XML Viewer for Tabular Forms for use with Mechanical Documentation," by Inoue, O., Tsuchida, K., Nakaghawa, S., Arita, T. and Yaku, T. which appeared in *Proc. IASTED AI 2003*, pp.1284–1289 (2003).

© 2012 Information Processing Society of Japan

Scalable systems can be constructed by using XML, and it is possible to offer diversified services. XML provides a basic syntax that can be used to share information between different kinds of computers, different applications, and different organizations on a global scale. Dejean et al. presented a system for converting PDF documents into a structured XML format [12]. Wang et al. proposed a new machine learning based approach for table detection from generic Web documents [13]. Anslow et al. demonstrated that an XML Data Storage Environment (XDSE) can be used to store program traces [14].

In our project, we have investigated the use of HIerarchical flow CHART description language (Hichart) [15], [16] for algorithms and data structure descriptions, and Hiform [17] for specifications. Hichart is a program flowchart description language that makes it easy to comprehend the program structure and makes it possible to describe a data structure. Hiform is a program specification language based on ISO6592 [18]. A Hiform document is a collection of tabular style templates and is represented by an attribute marked graph. In this paper, we consider Hiform as a tabular form document. Its syntactic structure is formulated by an attribute graph grammar with neighbourhood controlled embedding and dynamic edge relabeling (edNCE graph grammar, where d stands for "directed graphs") [19], [20], [21], and we developed a global creation and management system for Hiform based on this grammar. XML source codes of Hiform are automatically generated by parsing with the grammar. The XML source codes are automatically registered to a database. That is, we automate XML generation and registration process. Moreover, the specifications are retrieved from the database with a common Web browser. Our system can perform a characteristic retrieval for software development, for example, retrieval by a program name and a date of change. Hiform XML representation makes it easy to display on the Web, change the display style, exchange data and register to a database. A copious amount of data can be displayed in many forms by using XML. We perform parse and attribute evaluation in table processing using a unified formal methodology based on a graph model and graph grammar for tabular forms.

Tables are very popular diagrams in documents and have been used for descriptions of data lists, software interfaces of database systems (e.g., Ref. [22]), document layout, and so on. Watanabe et al. proposed a document recognition system for tables in 1995 [23]. This system distills letters and the table structure from scanned graphics. Amano et al. proposed a table form document analysis and synthesis system [24], [25]. This proposed system analyzes the table structure based on the string grammar, which is called the document structure grammar, and generates the synthesized documents. This system used a table form represented by a collection of boxes. In Ref. [26], [27], they presented a structure analysis based on a graph grammar. In addition, they proposed TFML [25], [27] based on XML, which contains structure and layout information. The basic structure of TFML reflects the indication pattern of the document. These approaches analyze images of tables and are not sufficiently applicable for the correct arrangement of item cells. Our approach is to formalize tables, and our grammar not only formalizes the graphical images of tables, but also determines the class in program documents. We construct our grammar in consideration of generating program documents and verifying structures of these documents. That is, this study recognizes the right arrangements of item cells in program documents. There have also been a variety of other studies concerning XML and databases. Ohata et al. proposed Java Alias Analysis Tool (JAAT) [28] including an XML database for storing analysis information and a useful GUI for the program maintainers. JAAT can analyze large programs or libraries such as the JDK class library and save internal syntactic and semantic information as an external XML database. The user interface subsystem has two main functions: editing programs and visualizing the results. Our study targeted a class of tabular forms: Hiform, that is, program documents. We formulate the syntactic structure of Hiform by means of an attribute edNCE graph grammar and automatically generate XML source codes with the correct Hiform structures via attribute evaluation with semantic rules for XML generation. Moreover, our system can register the XML documents with the XML and Hiform structures on the database and perform a characteristic retrieval for software specifications.

Our approaches can be applied to general tabular forms. For example, a great deal of data input and checking will be accomplished automatically. The efficiency of human labor is expected to be considerably improved by using a unified formal methodology. The method has enormous significance in terms of the cost of information processing. For example, it has a great potential to achieve complete automatic processing of paper documents such as ledger sheets and financial statements, and this is one possible application of this study being considered. Recently, with financial globalization, there has been a real need to find a faster technique for disclosing and circulating important information. Different formats are used in a variety of enterprises for processes such as financial information disclosure, auditing, tax preparation, reporting to relevant authorities, and analyzing information. Consequently, the flow of data between companies is becoming extremely difficult. Therefore, the development of an efficient data input system is prohibitively difficult. As a result, Since input from paper documents is done manually, a large amount of labor is required, and mistakes occur in the process. An electronic document law [29] came into force on April 1, 2005 in Japan. Although the number of companies publishing financial data as electronic data is increasing, automatic data processing is still not in full swing due to the different data formats used by different companies. eXtensible Business Reporting Language (XBRL) [30] is attracting attention as a possible solution to this problem, and applications corresponding to XBRL are expected to be developed and used at various companies. However, many companies have existing data in the form of paper documents or PDF format and; therefore, gearing them for XBRL has become an important problem. This study can be applied to the problem of creating a reasonable solution by defining a graph model and graph grammar for tabular forms, and the forms and contents of tables can be checked. Since the intended data can be extracted using an image recognition technique, it is possible to convert paper documents to XBRL formats, that is, a series of processes of converting can be formally systematized using our methodology

© 2012 Information Processing Society of Japan

ProgramName : Hanoi	General Document A1 p
Subtitle : Hanoi	
Library Code : cs-2002-02	Version : 1.0
Author : Yasunori Shiono	Original Release : 2002/01/21
Approver :	Current Release : 2002/01/31
Key Words : Hanoi Tower	CR Code :
Scope : Fundamental	
Variants :	
Language : Java	Software Req. : Java2 SDK 1.3
Operation : Interactive Batch RealTime( )	Hardware Req. :
References :	

Function :1. List and Explanation of Input Data or Parameters,
2. List and Explanation of Output Data or Result Values.

```
1. list and explanation of input data.
      int n; [Number of Plates]
      String target;      [Target Symbol]
      String work;        [Working Symbol]
      String destination; [Destination Symbol]

2. list and explanation of output data or result values.
      output data : No. to be moved: Source Symbol -> Destination Symbol
      return value: void
```

Example :

```
1. Example of Operation
      hanoi(5,A,B,C)

2. Example of Output
      1: A->C
      2: A->B
      1: C->B
      3: A->C
      1: B->A
      .......
```

Fig. 1 General document in Hiform.

and image recognition technique.

The rest of this paper is organized as follows. Section 2 explains Hiform documents and reviews graph grammar and parsing for Hiform. Section 3 describes automatic generation of XML files by attribute evaluation using the graph grammar. Section 4 presents a Hiform creation management system based on the graph grammar. Section 5 discusses our system, and Section 6 concludes the paper.

2. Definitions for Hiform

In this section, we initially explain Hiform documents [17], [18]. Next, we define its graph grammar [19], [20], [21] and parsing for Hiform [5], [19], [20], [21].

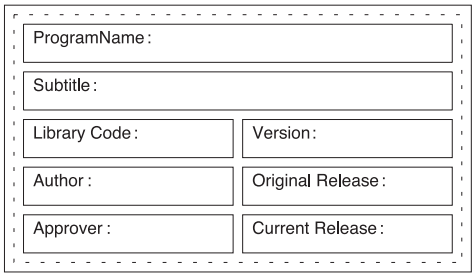
2.1 Hiform Documents [17], [18]

Hiform is a collection of tabular form specifications and includes all items defined in the ISO6592 guideline. Hiform is defined by 17 types of forms. Hiform documents include various items for software development. **Figure 1** shows a general document in Hiform.

2.2 Graph Grammar for Hiform [19], [20], [21]

Hiform is characterized by a graph grammar for graph syntax and attribute rules for drawing conditions. In the graph grammar of Hiform, a specification form is represented by a marked graph with a location. We illustrate an example in **Fig. 2**. This graph is constructed as follows.

(1) A node label of the graph shows an item in a tabular form.



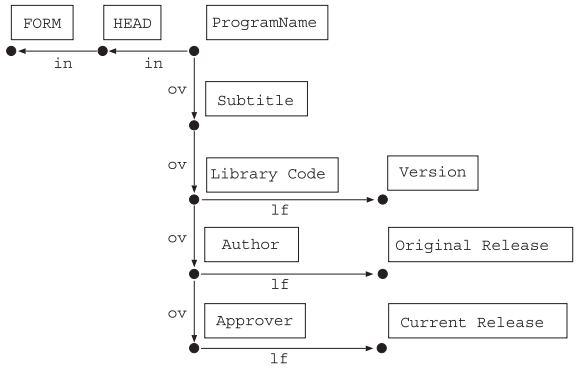


Fig. 2 Tabular form specification and corresponding marked graph.

(2) An edge label shows relations between items. “in” denotes “within,” “ov” denotes “over,” and “lf” denotes “left of.”

The graph grammar for Hiform is called *Hiform Nested Graph Grammar* (*HNGG*). This grammar formalizes an arrangement of items by productions and layout information for drawing tabular

© 2012 Information Processing Society of Japan

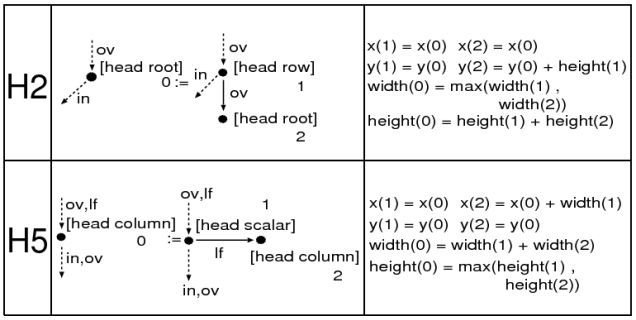


Fig. 3 Two productions with layout semantic rules.

Left \ Right	[head scalar]			[head column]			[head row]			[head root]		
	in	ov	lf	in	ov	lf	in	ov	lf	in	ov	lf
[head scalar]		<·>	<·		<·>	≐		<·>			<·>	
[head column]		<·>			<·>			<·>			<·>	
[head row]		<·			<·			<·			≐	

Fig. 4 Part of precedence relation of HNGG.

forms by attributes. HNGG is an attribute edNCE graph grammar and is defined as follows.

HNGG = $<G_N, A_N, F_N>$ generates marked graphs in Hiform. The underlying graph grammar $G_N = (\Sigma_N, \Delta_N, \Gamma_N, \Omega_N, P_N, S_N)$ is an edNCE context-free graph grammar, where Σ_N is the alphabet of node labels, $\Delta_N \subseteq \Sigma_N$ is the alphabet of terminal node labels, Γ_N is the alphabet of edge labels, $\Omega_N \subseteq \Gamma_N$ is the alphabet of final edge labels, P_N is the finite set of productions, and $S_N \in \Sigma_N - \Delta_N$ is the initial nonterminal. A production is of the form $X \to (D, C)$ with $X \in \Sigma_N - \Delta_N$, D is a graph over Σ_N and Γ_N, and $C \subseteq \Sigma_N \times \Gamma_N \times \Gamma_N \times V_D \times \{in, out\}$ is the connection relation, where V_D is a set of nodes on D. A_N is the finite set of attributes, and F_N is the finite set of semantic rules.

For example, **Fig. 3** illustrates two productions of HNGG. Each production has semantic rules for layout information. HNGG includes 280 productions and 1,248 semantic rules for the definition of marked graphs in Hiform.

2.3 Parsing for Hiform [5], [19], [20], [21]

HNGG has precedence relation for efficient parsing. We gave HNGG precedence relation based on Franck's precedence relation [5]. **Figure 4** shows a part of the precedence relation of HNGG. Precedence relations are determined by a connection relation of each production. Thus, every edge of a marked graph has exactly one of the precedence relations: <·, ≐, ·>, and <·>. We constructed 5,376 relations in HNGG, as shown in Fig. 4. The relations are shown to be pairwise disjoint. Thus, all precedence relations of HNGG are without conflict. Furthermore, all rules are uniquely invertible, and there is no reflexive nonterminal label in HNGG. Therefore, HNGG is a precedence attribute edNCE graph grammar. We use Frank's parsing algorithm because HNGG is a precedence graph grammar. This algorithm runs in linear time with respect to the number of nodes and edges in an input graph. Consequently the parsing algorithm of Hiform by HNGG is given by Franck's linear time parsing algorithm.

The parsing algorithm of Hiform by HNGG repeats Procedure Reduce until an input graph becomes the start graph, which is accomplished by Algorithm 1; otherwise, a production or a handle are not found. Algorithm Reduce searches for a handle in an input graph and repositions this handle to a new node that has a label on the left hand side of the production. The reducing algorithm is composed of four steps.

Algorithm 1 Parse (G)

Input: G : A marked graph.
Output: T : A derivation tree.

```
1: T is initialized; /* T is empty */
2: while G is not the start graph of HNGG do
3:     Reduce(G, T);
4: end while
5: return T
```

Procedure Reduce (G, T)

Step 1. A handle in G is searched based on the precedence relations of HNGG. If a handle is not found, this parsing algorithm is stopped.

Step 2. A production p: $X \to (D, C)$ of HNGG is searched, where D is isomorphic for the handle obtained in Step 1.

Step 3. The handle obtained in Step 1 is replaced by reverse application of p. By applying Step 3, G becomes a new graph.

Step 4. New nodes that are obtained from Step 3 are added into the derivation tree T.

3. Automatic Generation of XML Files by Attribute Evaluation [31]

Layout problems of tabular forms can be solved by attribute evaluation [5], [11], [19], [20], [21]. In HNGG, we use attributes *x*, *y*, *width* and *height* for layout. These attribute values are calculated using a derivation tree, which is output by the parsing algorithm of Hiform. In this section, we describe the XML representation of tabular forms and explain a new attribute S_{XML}, which contains XML source codes. This attribute S_{XML} is calculated using a derivation tree with layout attributes.

3.1 XML Representation of Tabular Forms

We explain the XML representation of tabular forms with an example of part of Hiform shown in **Fig. 5**. **Figure 6** shows the marked graph corresponding to the tabular form in Fig. 5. **Figure 7** is the XML element structure with node labels for the marked graph in Fig. 6. The root element is a ⟨graph⟩ element. Nodes of marked graphs are represented by ⟨node⟩ elements. A ⟨node⟩ element has attributes for a label and layout. A ⟨graph⟩ element has a ⟨node⟩ with label [FORM] as a child element. Descendant elements of the element ⟨node⟩ with label [FORM] is constructed as follows.

(1) If there is an "in"-labeled edge from the node v_1 to node v_2, the element ⟨node⟩ for v_1 is the child of element ⟨node⟩ for v_2.
(2) If there is a "lf"-labeled edge from node v_1 to node v_2, the element ⟨node⟩ for v_2 is the child of the element ⟨node⟩ for v_1.
(3) If there is an "ov"-labeled edge from node v_1 to node v_2, the element ⟨node⟩ for v_1 and the element ⟨node⟩ for v_2 are siblings.

Therefore, as shown in Fig. 7, the ⟨node⟩ with label [FORM] has

© 2012 Information Processing Society of Japan

ProgramName :	
Subtitle :	
Library Code :	Version :
Author :	Original Release :
Approver :	Current Release :

Fig. 5 Part of tabular form specification.

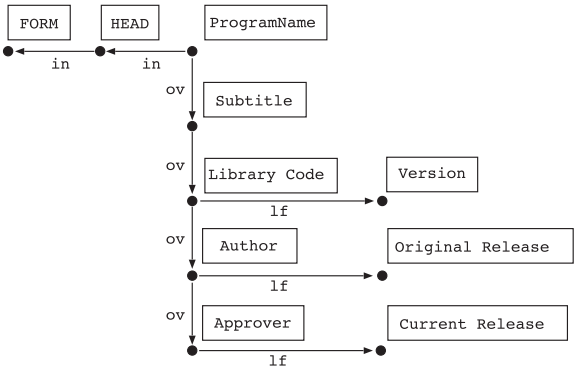


Fig. 6 Marked graph corresponding to tabular form of Fig. 5.

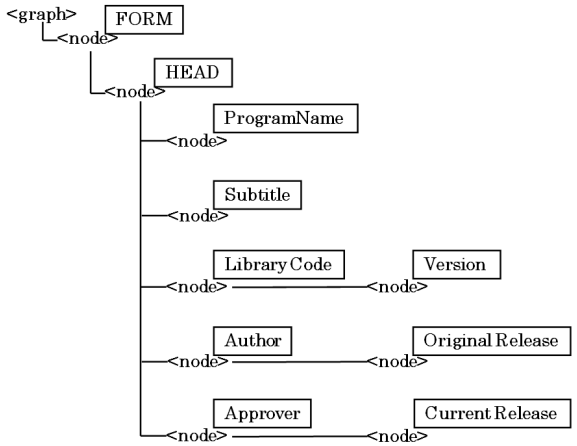


Fig. 7 XML element structure with node labels for marked graph of Fig. 6.

an element ⟨node⟩ with label [ProgramName] as a child element. The elements ⟨node⟩ with labels [ProgramName], [Subtitle], [Library Code], [Author], and [Approver] are siblings, and the elements ⟨node⟩ with labels [Library Code], [Author], and [Approver] have ⟨node⟩ with labels [Version], [Original Release], and [Current Release], respectively. **Figure 8** is the XML source code corresponding to the marked graph of Fig. 6.

3.2 Attribute Definition and Evaluation for XML

The attribute S_{XML} for XML is computed by referring to other attributes and using a concatenation operator. The XML source codes are generated by evaluating S_{XML}. **Figure 9** shows a process flow of the XML generation.

First, a derivation tree is generated from a marked graph by parsing using HNGG. Next, layout attributes *x*, *y*, *width* and *height* are computed by layout attribute evaluation using layout semantic rules, and a derivation tree with the layout information

```
<graph>
 <node label="FORM" x="0" y="0" width="200" height="150">
  <node label="HEAD" x="0" y="0" width="200" height="150">
   <node label="Program Name" x="0" y="0" width="200" height="30">
   </node>
   <node label="Subtitle" x="0" y="30" width="200" height="30">
   </node>
   <node label="Library Code" x="0" y="60" width="100" height="30">
    <node label="Version" x="100" y="60" width="100" height="30">
    </node>
   </node>
   <node label="Author" x="0" y="90" width="100" height="30">
    <node label="Original Release" x="100" y="90" width="100" height="30">
    </node>
   </node>
   <node label="Approver" x="0" y="120" width="100" height="30">
    <node label="Current Release" x="100" y="120" width="100" height="30">
    </node>
   </node>
  </node>
 </node>
</graph>
```

Fig. 8 XML source code corresponding to marked graph of Fig. 6.

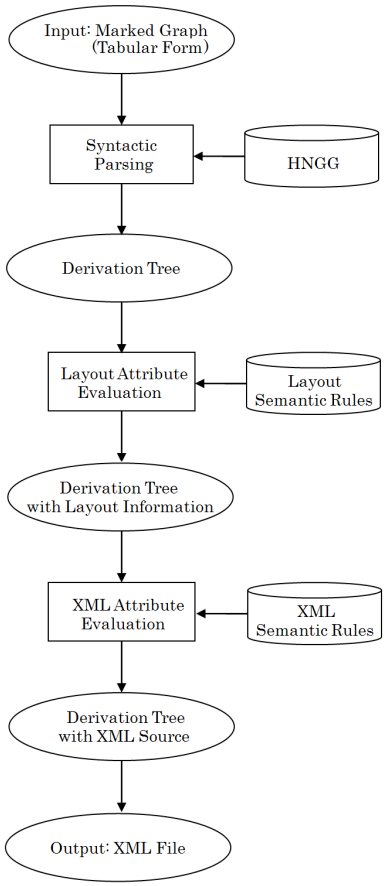


Fig. 9 Process flow of XML generation.

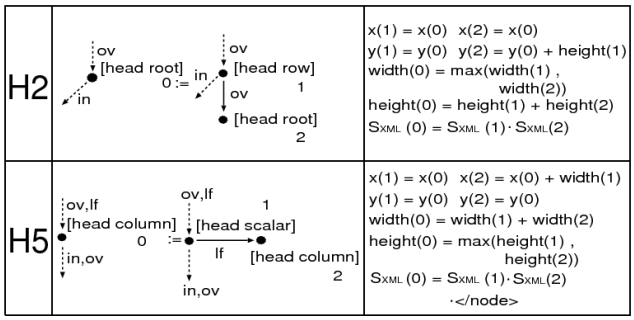


Fig. 10 Two productions with layout semantic rules.

is obtained. The attribute S_{XML} is computed on the derivation tree with layout attributes by XML attribute evaluation using XML semantic rules. There are 280 semantic rules for XML. For example, **Fig. 10** illustrates two productions with XML semantic

© 2012 Information Processing Society of Japan

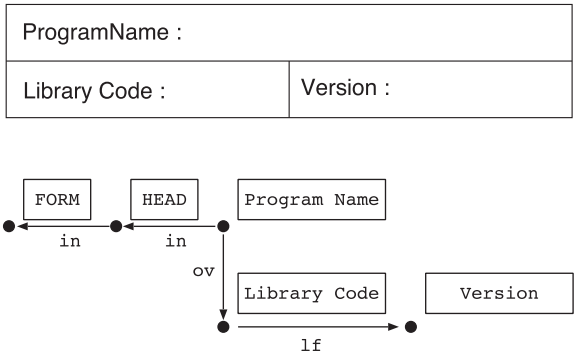


Fig. 11 Simple tabular form T1 and it's marked graph.

rules.

Attribute evaluation is performed in a bottom-up manner. An algorithm for XML attribute evaluation S_{XML}-Evaluate is as follows.

Algorithm 2 S_{XML}-Evaluate (T_{layout})

Input: T_{layout} : A derivation tree with layout information
Output: T_{xml} : A derivation tree with XML source
/* v : a node of T_{xml}. */
1: $T_{xml} \leftarrow T_{layout}$;
2: $v \leftarrow$ the root node of T_{xml};
3: Node-S_{XML}-Evaluate (v);
4: **return** T_{xml}

Procedure Node-S_{XML}-Evaluate (v)
/* v_c : a child node of v. */
1: **if** v has nonterminal label **then**
2: **for** each child node v_c **do**
3: Node-S_{XML}-Evaluate (v_c)
4: **end for**
5: Evaluate $S_{XML}(v)$
6: **end if**

Theorem 1 *The time complexity of the algorithm S_{XML}-Evaluate is $O(n)$, where n is the number of nodes in a derivation tree.*

Proof. All nodes in a derivation tree are handled by the procedure Node-S_{XML}-Evaluate. The time complexity of the algorithm Evaluate S_{XML} is $O(n)$. Hence, the time complexity of the algorithm is $O(n)$. □

Finally, S_{XML} of the root node in a derivation tree with layout attributes is the XML source code.

We explain the process of generating from the tabular form T1 in **Fig. 11** to the XML source as an example. Figure 11 is the tabular form T1 and it's marked graph. The tabular form T1 is a simple explanatory tabular form.

First, a derivation tree is generated from the marked graph of T1 by parsing using HNGG, and layout attributes x, y, *width*, and *height* are computed by layout attribute evaluation using layout semantic rules. **Figure 12** is the obtained derivation tree with layout attributes. In Fig. 12, closed circles are terminal nodes, boxes are nonterminal nodes, and strings in square brackets or a square near nodes are node labels. Numbers adjacent to node labels are node IDs. x, y, w, and h are attributes x, y, *width*, and *height*, respectively. Next, the attribute S_{XML} is computed from the derivation tree with layout attributes by using the algorithm S_{XML}-Evaluate. The evaluation of attributes is performed in a bottom-up manner. The handling procedure is as follows. In this procedure, the numbers are node IDs, for example, v_0 is the node of ID 0.

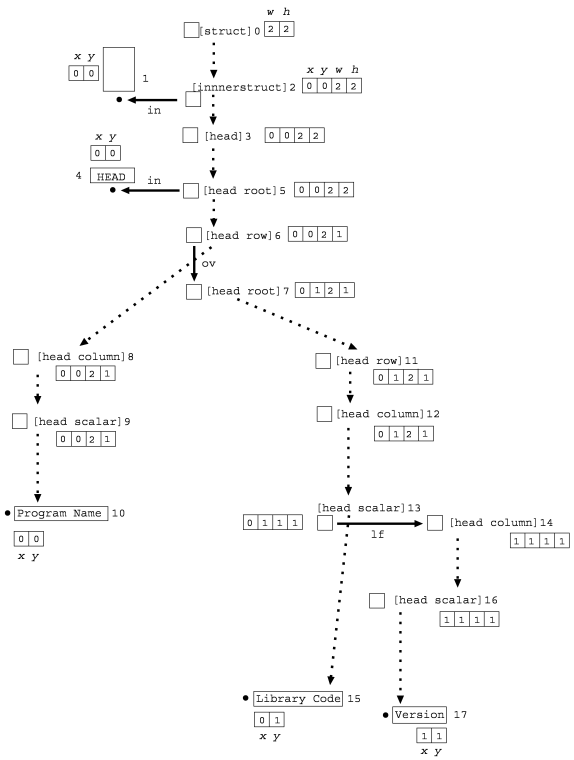


Fig. 12 Derivation tree with layout attributes of T1.

1: Since v_0 has an unhandled child node, handle v_1
2: Since v_1 has a terminal label, v_1 is not evaluated
3: Since v_0 has an unhandled child node, handle v_2
4: Since v_2 has an unhandled child node, handle v_3
5: Since v_3 has an unhandled child node, handle v_4
6: Since v_4 has a terminal label, v_4 is not evaluated.
7: Since v_3 has an unhandled child node, handle v_5
8: Since v_5 has an unhandled child node, handle v_6
9: Since v_6 has an unhandled child node, handle v_8
10: Since v_8 has an unhandled child node, handle v_9
11: Since v_9 has an unhandled child node, handle v_{10}
12: Since v_{10} has terminal label, v_{10} is not evaluated
13: Since v_9 has no unhandled child node, evaluate $S_{XML}(v_9)$
14: Since v_8 has no unhandled child node, evaluate $S_{XML}(v_8)$
15: Since v_6 has no unhandled child node, evaluate $S_{XML}(v_6)$
16: Since v_5 has an unhandled child node, handle v_7
17: Since v_7 has an unhandled child node, handle v_{11}
18: Since v_{11} has an unhandled child node, handle v_{12}
19: Since v_{12} has an unhandled child node, handle v_{13}
20: Since v_{13} has an unhandled child node, handle v_{15}
21: Since v_{15} has a terminal label, v_{15} is not evaluated
22: Since v_{13} has no unhandled child node, evaluate $S_{XML}(v_{13})$
23: Since v_{12} has an unhandled child node, handle v_{14}
24: Since v_{14} has an unhandled child node, handle v_{16}
25: Since v_{16} has an unhandled child node, handle v_{17}

© 2012 Information Processing Society of Japan

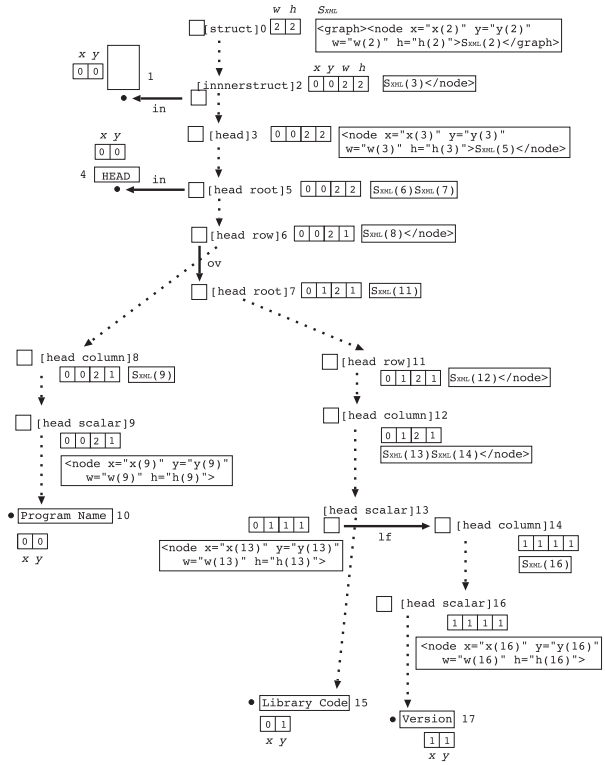


Fig. 13 Derivation tree with XML source code of T1.

```
<graph>
 <node label="FORM" x="0" y="0" width="2" height="2">
  <node label="HEAD" x="0" y="0" width="2" height="2">
   <node label="Program Name" x="0" y="0" width="2" height="1">
   </node>
   <node label="Library Code" x="0" y="1" width="1" height="1">
    <node label="Version" x="1" y="1" width="1" height="1">
    </node>
   </node>
  </node>
 </node>
</graph>
```

Fig. 14 XML source code of tabular form T1.

26: Since v_{17} has a terminal label, v_{17} is not evaluated
27: Since v_{16} has no unhandled child node, evaluate $S_{XML}(v_{16})$
28: Since v_{14} has no unhandled child node, evaluate $S_{XML}(v_{14})$
29: Since v_{12} has no unhandled child node, evaluate $S_{XML}(v_{12})$
30: Since v_{11} has no unhandled child node, evaluate $S_{XML}(v_{11})$
31: Since v_7 has no unhandled child node, evaluate $S_{XML}(v_7)$
32: Since v_5 has no unhandled child node, evaluate $S_{XML}(v_5)$
33: Since v_3 has no unhandled child node, evaluate $S_{XML}(v_3)$
34: Since v_2 has no unhandled child node, evaluate $S_{XML}(v_2)$
35: Since v_0 has no unhandled child node, evaluate $S_{XML}(v_0)$

Therefore, nodes of IDs 9, 8, 6, 13, 16, 14, 12, 11, 7, 5, 3, 2, and 0 are evaluated for S_{XML} in that order. AS a result, the derivation tree with XML source code is obtained. **Figure 13** is the obtained derivation tree with XML source code. The evaluation results of S_{XML} are shown in Fig. 13. Finally, S_{XML} of the root node is the XML source code for tabular form T1. **Figure 14** shows the obtained XML source code of T1.

Generated XML files can be browsed by applying the eXtensible Stylesheet Language Transformations (XSLT) stylsheet to them. **Figure 15** shows part of the XSLT stylesheet for browsing XML files of Hiform. After the XSLT stylesheet is applied, an XML file of Hiform is converted into an HTML document. The result of displaying an XML file of T1 with Internet Explorer is shown in **Fig. 16**.

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/TR/WD-xsl">

<xsl:template match="/">
 <html>
 <head>
  <title></title>
 </head>

 <body>
  <div>
   <xsl:apply-templates select="document/graph/node" />
  </div>
 </body>
 </html>
</xsl:template>

<xsl:template match="document/graph/node">
 <xsl:apply-templates select="node[@label='HEAD']" />
 <xsl:apply-templates select="node[@label='A1']" />
</xsl:template>
   :
   :
   :
   :
   :
   :
<xsl:for-each select="node">
 <td valign="CENTER">

  <xsl:if test="@width[.= 100]">
   <xsl:attribute name="width">
   400
   </xsl:attribute>
  </xsl:if>

  <xsl:if test="@width[.= 200]">
   <xsl:attribute name="colspan">
   2
   </xsl:attribute>
  </xsl:if>

  <xsl:value-of select="@label" />

 </td>
</xsl:for-each>

</tr>
</xsl:for-each>
</table>
</xsl:template>

</xsl:stylesheet>
```

Fig. 15 Part of XSLT stylesheet file.

Fig. 16 Display screen of T1 using Internet Explorer.

4. Hiform Creation Management System

Since the syntactic structure of Hiform is formulated by HNGG that is a precedence attribute edNCE graph grammar, we developed a Hiform creation management system based on HNGG. **Figure 17** illustrates the structure of the system. The system consists of a Hiform editor for Hiform creation and its XML generation, a database registration system for XML database registration, and a database interrogation system for Hiform interrogation. The XML files can be browsed by applying the XSLT stylesheet to them. The following are explanations of the three systems.

4.1 Hiform Editor

The Hiform editor has a graph parsing engine, which consists of the following parts.

(1) Productions of HNGG.

© 2012 Information Processing Society of Japan

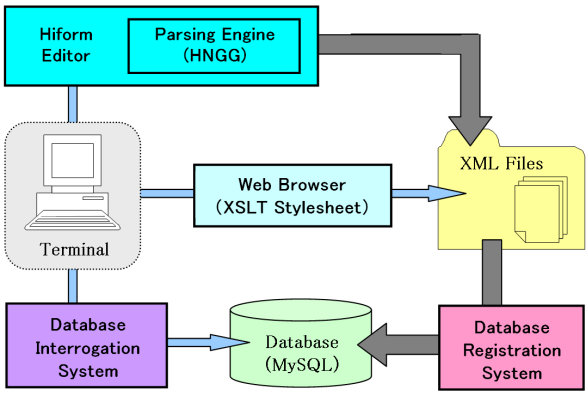


Fig. 17 System structure of Hiform creation management system.

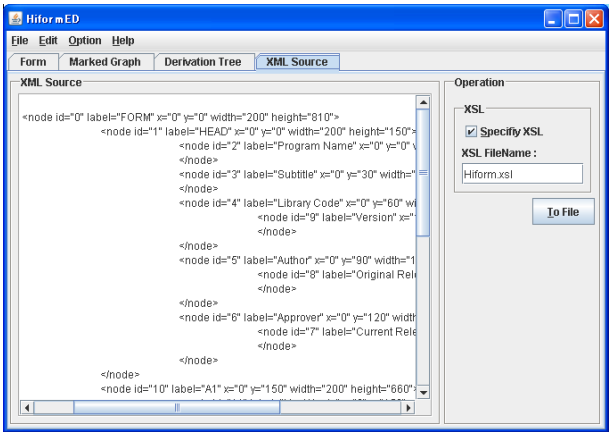


Fig. 18 Execution screen of Hiform editor.

(2) Precedence relation table for syntactic parsing.
(3) Semantic rules for layout and XML.

Figure 18 is an execution screen of the Hiform editor. Users can create Hiform and generate its XML file. Syntactic parsing and attribute evaluation are performed in the process flow shown in Fig. 9, and the XML file can be generated. Although a part of Hiform can be created and inner data can be displayed, dialogical editing is not currently implemented.

4.2 Database Registration System

The database registration system registers XML files of Hiform with XML file structures on the MySGL relational database. Since the XML files are automatically registered after parsing, our system checks whether the XML files assort the Hiform format. The system uses a part of the free PHP library PXBASE [32] for XML file registration. The database keeps DOM tree structures. Therefore, registered data can be dealt with using MySQL commands.

4.3 Database Interrogation System

The database interrogation system is a system with which users can browse and retrieve XML documents of Hiform on a database by using a common Web browser. Additionally, users can view history information. The system has the following main features.

(1) The page is divided into condition input and search result parts by frames.
(2) Users can search XML files of Hiform by project name, program name, personal name, or date of issue.
(3) The search results are displayed by one of four display forms: header, table, personal name, or project name.
(4) Users can view, register, and change the history.
(5) Users can directly access XML files of Hiform, which are displayed by a form of Hiform.

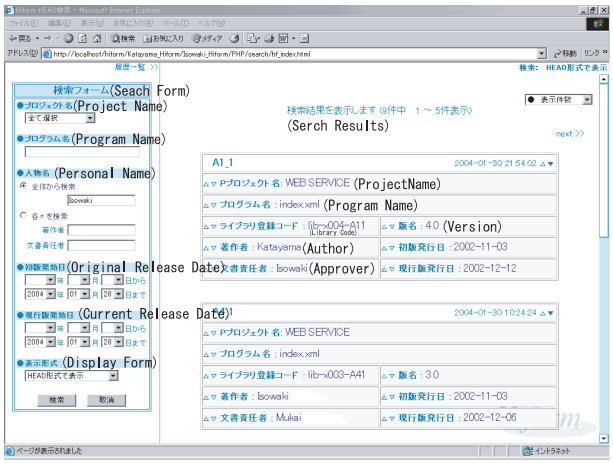


Fig. 19 Display of search results in header form.

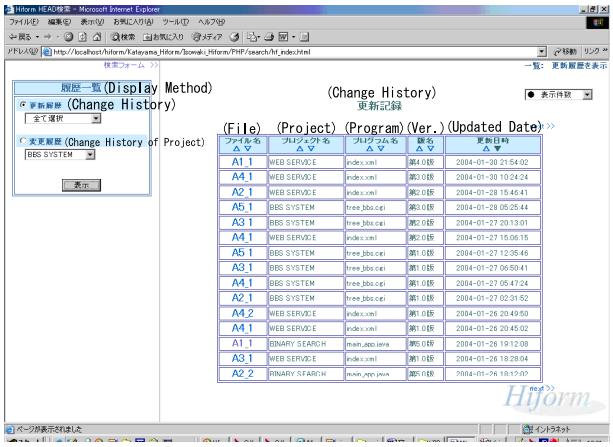


Fig. 20 Display of change history.

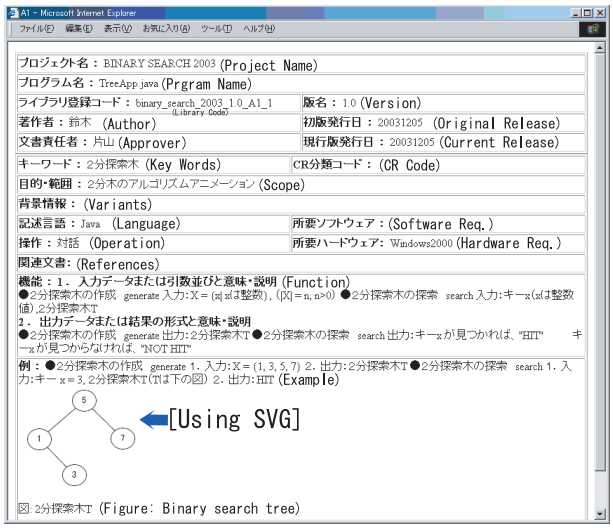


Fig. 21 Display of searched XML file of Hiform.

Figures 19 and **20** are display screens of search results in the

© 2012 Information Processing Society of Japan

header form and change history, respectively. **Figure 21** shows a display screen of a searched XML file of Hiform. These results are browsed by Internet Explorer.

5. Discussion

Our system parses a given Hiform specification by using HNGG for Hiform and generates a derivation tree. Software information is extracted from the structure and contents of each node of the derivation tree. The system can evaluate various attribute values such as important item of software information, developer, release date, development language, and layout. Finally, the results are automatically output as XML files. Therefore, since the XML files can be displayed with a common Web browser by defining the stylesheet, anyone can easily view the results.

The XML files are automatically registered with the XML file structure on the relational database, and the specifications are retrieved from the database. Our system can perform a characteristic retrieval for software specifications, for example, retrieval by a program name and a date of change. A specification may be changed several times in software development. At that time, a user can interrogate specifications by specifying a particular period on the search form shown in Fig. 19. Change history can also be displayed, as shown in Fig. 20. Moreover, change history list of a project can be displayed, as shown in **Fig. 22**.

We developed the system based on a formal methodology and the XML files are registered with the XML file structures. Therefore, although we do not implement on the system, there are feasible effective functions, for example, retrieval of Unified Modeling Language (UML).

The diagram of the specification in Fig. 21 is drawn using Scalable Vector Graphics (SVG). Since the XML files are registered with the XML file structure on the database, it is possible to retrieve the specifications that have figures: the node has number symbols and the UML class diagram is connected to particular classes. In this way, users can retrieve and exchange software specifications including text and figures on the Web, and the specifications can be displayed and printed by common Web browsers.

In another instance, since the process from input to registration on the database using graph grammar is automatically performed, tabular forms can be standardized. For example, although the items of the date and project name may be written anywhere, documents can be checked by syntactic parsing for global diagrammatic structures, whether the date and project name are correctly written at the beginning of documents in that order. In this way, the layout can be standardized using a formal methodology. Although it is difficult to check by XML and Document Type Definition (DTD), formulation using graph grammar can achieve form standardization.

Our approaches can be applied to formal tabular form processing system for tabular form documents. The efficiency of human labor is expected to be considerably improved by using a unified formal methodology based on graph theory and advanced retrieval. The methodology has enormous significance in terms of the cost of information processing.

(Change History of Project)
変更履歴
(BINARY SEARCH project)(Project Name)

(Ver.)	第1版	第2版	第3版	第4版	第5版
DATE	2003-05-26	2003-06-03	2003-06-06	2003-07-19	2003-09-19
A1_1	○	○			○
A2_1	○		○	○	
A2_2	○			○	○
A3_1	○		○		
A4_1	○	○			
A5_1	○	○			

Fig. 22 Display of change history list in project.

6. Conclusion

We defined an attribute for XML representation in Hiform graph grammar HNGG and developed a Hiform creation management system based on HNGG. The XML files of Hiform are automatically generated by attribute evaluation using the XML semantic rules and are registered with the XML file structures on the relational database. We automated the process from creation to registration. Moreover, our system can perform a characteristic retrieval for software specifications.

Future work is to achieve the characteristic retrieval and standardization we have described in Section 5 and to enable automatic processing of paper documents by using scanners.

References

[1] Lopresti, D. and Nagy, G.: A Tabular Survey of Automated Table Processing, *Lecture Notes in Computer Science 1941*, pp.93–120 (2000).

[2] Zanibbi, R., Blostein, D. and Cordy, J.: A Survey of Table Recognition: Models, Observations, Transformations and Inferences, *International Journal on Document Analysis and Recognition*, Vol.7, No.1, pp.1–16 (2004).

[3] Embley, D., Hurst, D., Lopresti, D. and Nagy, G.: Table-processing Paradigms: A research survey, *International Journal on Document Analysis and Recognition*, Vol.8, No.2-3, pp.66–86 (2006).

[4] Teitelbaum, T. and Reps, T.: The Cornell Program Synthesizer: A Syntax-Directed Programming Environment, *Comm. ACM*, Vol.24, No.9, pp.563–573 (1981).

[5] Franck, R.: A Class of Linearly Parsable Graph Grammars, *Acta Infomatica*, Vol.10, pp.175–201 (1978).

[6] Vigna, P.D. and Ghezzi, C.: Context-free graph grammars, *Inf. Control*, Vol.37, No.2, pp.207–233 (1978).

[7] Rozenberg, G. (Ed.): *Handbook of Graph Grammar and Computing by Graph Transformation, Volume 1 Foundations*, World Scientific (1997).

[8] Ehrig, H., Engels, G., Kreowski, H.-J. and Rozenberg, G. (Eds.): *Handbook of Graph Grammar and Computing by Graph Transformation, Volume 2 Aplications, Lnguags and Tools*, World Scientific (1997).

[9] Nishino, T.: Attribute Graph Grammars with Applications to Hichart Program Chart Editors, *Advances in Software Science and Technology*, Vol.1, pp.426–433 (1989).

[10] Adachi, Y., Anzai, K., Tsuchida, K. and Yaku, T.: Hierarchical Program Diagram Editor Based on Attribute Graph Grammar, *Proc. 20th Conference on Computer Software and Applications Conf.*, pp.205–213 (1996).

[11] Adachi, A., Tsuchida, T. and Yaku, T.: Program visualization using attribute graph grammars, *Proc. 15th IFIP World Computer Congress 98* (1998).

[12] Dejean, H. and Meunier, J.L.: A System for Converting PDF Documents into Structured XML Format, *Lecture Notes in Computer Science*, Vol.3872, pp.129–140 (2006).

[13] Wang, Y. and Hu, J.: Detecting Tables in HTML Documents, *Lecture Notes in Computer Science*, Vol.2423, pp.249–260 (2002).

[14] Anslow, C., Marshall, S., Biddle, R., Noble, J. and Jackson, K.: XML Database Support for Program Trace Visualisation, *Proc. 2004 Aus-*

© 2012 Information Processing Society of Japan

tralasian Symposium on Information Visualisation, Vol.35, pp.25–34 (2004).
[15] Sugita, K., Adachi, A., Miyadera, Y., Tsuchida, K. and Yaku, T.: A visual programming environment based on graph grammars and tidy graph drawing, *Proc. ICSE '98*, Vol.II, pp.74–79 (1998).
[16] Goto, T., Kirishima, T., Motousu, N., Tsuchida, K. and Yaku, T.: A visual software development environment basedon graph grammars, *Proc. IASTED Software Engineering 2004*, pp.620–624 (2004).
[17] Sugita, K. and Yaku, T.: Hiform Reference Page (1999), available from ⟨http://www.sm.u-tokai.ac.jp/˜sugita/Hiform⟩.
[18] ISO6592-1985, Guidelines for the documentation of computer-based application systems (1985).
[19] Arita, T., Tomiyama, K., Yaku, T., Miyadera, Y., Sugita, K. and Tsuchida, K.: Syntactic Processing of Diagrams by Graph Grammars, *Proc. IFIP WCC ICS 2000*, pp.145–151 (2000).
[20] Arita, T., Sugita, K., Tsuchida, K. and Yaku, T.: Syntactic Tabular Form Processing By Precedence Attribute Graph Grammar, *Proc. IASTED AI 2001*, pp.637–642 (2001).
[21] Arita, T., Tomiyama, K., Tsuchida, K. and Yaku, T.: Application of Attribute NCE Graph Grammars to Syntactic Editing of Tabular Forms, *Electric Notes in Theoretical Computer Science*, Vol.50, Issue 3 (2001).
[22] Santucci, G. and Tarantino, L.: A Hypertabular Visualizer of Query Results, *Proc. 1997 IEEE Symposium on Visual Languages*, pp.193–200 (1997).
[23] Watanabe, T., Luo, Q. and Sugie, N.: Layout recognition of multi-kinds of table-form documents, *IEEE Trans. Pattern Analysis and Machine Intelligence*, Vol.17, No.4, pp.432–445 (1995).
[24] Amano, A., Asada, N., Motoyama, T., Sumiyoshi, T. and Suzuki, K.: Table Form Document Synthesis by Grammar-Based Structure Analysis, *Proc. 6th International Conference on Document Analysis and Recognition*, pp.533–537 (2001).
[25] Amano, A., Asada, N., Mukunoki, M. and Aoyama, M.: Table form document analysis based on the document structure grammar, *International Journal on Document Analysis and Recognition*, Vol.8, No.2-3, pp.201–213 (2006).
[26] Amano, A. and Asada, N.: Complex Table Form Analsis Using Graph Grammar, *Lecture Notes in Computer Science*, Vol.2423, pp.283–286 (2002).
[27] Amano, A. and Asada, N.: Graph Grammar Based Analysis System of Complex Table Form Document, *Proc. 7th International Conference on Document Analysis and Recognition*, pp.916–920 (2003).
[28] Ohata, F. and Inoue, K.: JAAT: Java Alias Analysis Tool for Program Maintenance Activities, *Proc. 9th IEEE International Symposium on Object and Component-oriented Real-time Distributed Computing* (*ISORC2006*), pp.232–242 (2006).
[29] Law Governing the Use of Information and Communications Technology in the Preservation of Documents that Private Businesses Perform, available from ⟨http://www.kantei.go.jp/foreign/policy/it/051031/law.pdf⟩.
[30] XBRL INTERNATIONAL, available from ⟨http://www.xbrl.org/Home/⟩.
[31] Inoue, O., Tsuchida, K., Nakagawa, S., Arita, T. and Yaku, T.: An XML Viewer for Tabular Forms for use with Mechanical Documentation, *Proc. IASTED AI 2003*, pp.1284–1289 (2003).
[32] Tanaka, H.: XML First Step PXBASE (MySQL), available from ⟨http://www.geocities.jp/xmlfirststep/mxbase/mxbase_menu.html⟩ (in Japanese).

Yasunori Shiono received his M.E. and Dr.Eng. degrees from Toyo University in 2006 and 2010 respectively. He is currently an Assistant Professor of Faculty of Information Sciences and Arts at Toyo University. His research interests include graph algorithms, graph grammars, fuzzy theory and software development environments. He is a member of IEICE Japan, JSSST, JSIAM and IEEE.

Tomokazu Arita received his M.S. and D.S. degrees from Nihon University in 2000 and 2009, respectively. He has been an Assistant Professor at J.F. Oberlin University since 2004. His research interests include graph languages, graph algorithms, and their applications.

Youzou Miyadera received his B.Sc., M.Sc. and D.Sc. degrees in engineering science from Tokyo Denki University in 1984, 1986 and 1998, respectively. He was on the Department of Information Sciences at Tokyo Denki University as an Instructor from April 1986 to March 1997. He has been on the department of Mathematics and Information Science at Tokyo Gakugei University as an Associate Professor until 2008. He is a professor of the division of Natural Science at Tokyo Gakugei University now. His current research interests include information visualization, programming language education environments and program analysis. He is a member of IEEE Computer Society, ACM, IPSJ, IEICE, Japan Society for Information and Systems in Education, and the Japan Society for Software Science and Technology.

Kimio Sugita received his M.Sc. from the University of Tokyo in 1968. His research interests include automaton, graph languages, and information processing education. He is a member of IEICE Japan, Mathematical Society of Japan, and American Mathematical Society.

Takeo Yaku received his M.Sc. and D.Sc. from Waseda University in 1972 and 1977, respectively. He has been a Professor at the Department of Computer Science and System Analysis of Nihon University since 1992. His research interests include software visualization, human interface, graph languages, and graph algorithms. He is a member of IEICE Japan, IEEE Computer Society and ACM.

© 2012 Information Processing Society of Japan

Kensei Tsuchida received his M.S. and D.S. degrees in mathematics from Waseda University in 1984 and 1994 respectively. He was a member of the Software Engineering Development Laboratory, NEC Corporation in 1984–1990. From 1990 to 1992, he was a Research Associate of the Department of Industrial Engineering and Management at Kanagawa University. In 1992 he joined Toyo University, where he was an Instructor until 1995 and an Associate Professor from 1995 to 2002 and a Professor from 2002 to 2009 at the Department of Information and Computer Sciences, and since 2009 he has been a Professor of Faculty of Information Sciences and Arts. He was a Visiting Associate Professor of the Department of Computer Science at Oregon State University from 1997 to 1998. His research interests include software visualization, human interface, graph languages, and graph algorithms. He is a member of IPSJ, IEICE Japan and IEEE Computer Society.

© 2012 Information Processing Society of Japan