File Structures for Modular Form Processing Systems

by
Shun-ichi NAKAGAWA

A Thesis Submitted to
Earth Information Mathematical Sciences,
Graduate School of Integrated Basic Sciences,
Nihon University
in Fulfillment of the Requirements
of the Degree of Master of Science

March, 2002

Abstract

A detailed description of software systems by program specification documents is very important to manage those systems. Tabular forms for program specification documents are formalized by a graph grammar for automatic processing of detailed descriptions. However, standardized expression for handling tabular forms is not yet established. In this paper, we propose a universal code (which we call FXL) for tabular forms. FXL has following characteristics:

(1) Syntax of FXL is defined by extended BNF. Therefore, codes of FXL can be syntactically verified. (2) Codes of FXL are text-based codes. Therefore, they can be edited directly. (3) FXL can describe several attributes for tabular forms, which are locations and positions of cells and geometrical relations among cells.

In this paper, we deal with a processing system of tabular form and a universal format of tabular forms for those processing system.

In Section 2 [17], we introduce methods and systems concerning to Hichart program diagrams and Hiform program documentation.

In Section 3 [17, 19], we consider the whole structure of tabular form documentation system. First we design the system structure of the system. Then, we design the file structure of the system. We note that ceratain intermidiate language of tabular form is necessary for data exchange among tabular form systems.

In Section 4, we deal with above intermediate language of tabular forms. We first provide the concept of the intermidiate language [17]. Next, we provide the specificatoin of the intermidiate language. We note that the class of tabular forms described by the intermediate language is proper super class of the class of tabular forms generated by Hiform graph grammar HNGG [13, 18].

In Appendix, we provide a specification manual of the intermidiate language.

Keywords software documentation, visualization, graph grammars, tabular forms.

Acknowledgments

I would like to express my greatest gratitudes for Professor Takeo YAKU at Nihon University, who has been encouraging and advising me throughout my whole study to complete this thesis.

I would also like to express thanks to Professor Kimio SUGITA at Tokai University, Professor Kensei TSUCHIDA at Toyo University, and Professor Youzou MIYADERA at Tokyo Gakugei University for their encouragements and valuable suggestions.

I would also like to thank Tomokazu ARITA at Nihon Univerity, for I could have fruitful discussions with him.

Shun-ichi NAKAGAWA

Earth Information Mathematical Sciences,
Graduate School of
Integrated Basic Sciences,
Nihon University
3-25-40 Sakura-jyosui, Setagaya-ku,
Tokyo 156-8550, JAPAN

Contents

Chapter 1

Introduction

Generally, software documentation includes program specification documents and program structure diagrams. We deal with program specification documents with tabular forms. So, it is necessary to develop automatic drawing and editing mechanism of them. This paper deals with general tabular forms and their mechanical manipulation problems.

Both a syntactic definition of tabular forms and a definition for drawing them are necessary for mechanical manipulation of them. Here, attribute graph grammars formulate syntactic structure of tabular forms. These grammars also formulate visual structures among cells in each form. Franck [1] introduced precedence graph grammars and applied them to nested diagrams called PLAN2D. Applications based on some graph grammars are proposed such as DiaGen[15], GenGed [16] and so on. In [8], table representations are proposed. We formulated the hierarchical structured diagram [6, 12].

Recently, XML [21] is recognized as one of the most standards concerning data exchange and web–based systems. Since XML is platform–independent, software documents expressed in XML are viewed and displayed on any readily available Web browsers. We are proposing and constructing an XML system for program documents [19]. This system generates XML files for program documents. These XML files are able to be displayed by combining special XSL files on XML viewers or Web browsers such as Internet Explorer.

In the 1980s, Hichart, PAD, SPD, and HCP were proposed as research of program diagrams. And H-code2 of list form etc was proposed as internal code for program diagrams. In 1995, DXL [5] was proposed as a universal code of them and defined by BNF.

In 2000, we introduced partly a syntactic definition of program specification forms based on ISO6592 standard [13, 18]. We employed graph grammars for formalizing those forms in [13]. In this paper, we propose a universal processing system for tabular forms. Accordingly, we employ, as universal models, attribute NCE graph grammars.

This thesis is organized as follows:
In Chapter 2, we review tabular forms for program specifications and a formal syntax of those forms based on an attribute NCE graph grammar. We introduce a parsing engine based on a graph grammar.

In Chapter 3, we introduce a syntactic processing system and the file structures using a parser for tabular forms, which provides mechanical verifier and drawer.

In Chapter 4, we propose a file format for our tabular form processing system.

In Chapter 5, we summarize our reults.

In Appendix, we report a detailed description of the file format.

Chapter 2

Known Results

In software development, description of its system structure and algorithms is very important. We review tabular forms for describing program specification concerning system development and management in this section. Furthermore, we also review a mechanism for modeling tabular forms and a system for analyzing those forms.

2.1 Tabular Forms for Program Specification

We consider here a program specification language called *Hiform* [9] based on ISO6592 [2].

The International Organization for Standardization issued a guideline in ISO6592 and described all items in program documentation in Annexes A, B and C. Sugita et al considered the ISO6592 items and introduced Hiform, which includes all items defined in these Annexes. Hiform [11] is defined by 17 types of forms. Figure 1 shows a Hiform program specification form. The order among tabular forms was already defined by a context-free string grammar (cf. [9]).

An arrangement of all items in a Hiform document and drawing parameters of its document are defined based on an attribute graph grammar. This grammar is called *Hiform Nested tabular form Graph Grammar* (HNGG). A

Program Name:	
Subtitle:	
Library Code:	Version:
Author:	Original Release:
Approver:	Current Release:
Problem Description:	
Problem Supplementary Information (Theoretical Principles, Methods and References):	
Problem Solution: 1.Conventions and Terminology 2.Principles and Algorithms	

Figure 1: An Example of Hiform

Hiform document is represented by a graph as Figure 2. A graph denotes a arrangement of all items in a document. Information for drawing a form is obtained from values of attributes with each item by analyzing a graph for it based on HNGG.

A *marked graph* as in Figure 2 is defined as follows. A *node label* of the graph shows an *item* of a tabular form. A node label called a *mark*. An *edge label* shows relations between items. 'lf' denotes the meaning of 'left of'. 'ov' denotes the meaning of 'over'. 'in' denotes the meaning of 'within'.

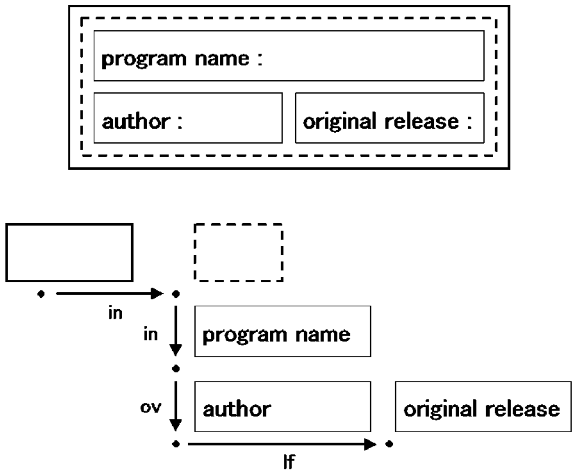

Figure 2: Nested tabular form and its corresponding marked graph

2.2 Parsing Engine

Our parsing engine is constructed on two parts, which are syntax analysis and attribute evaluation. Input of this parsing engine is a marked graph, and output is a derivation tree with attribute. In this part, we explain an abstract parsing process.

First, by syntax analysis for a marked graph with attribute, a derivation tree is generated. Next, by attribute evaluation for the derivation tree, an attribute derivation tree is generated. The attribute derivation tree has layout information. Tabular form is generated with this flow by a browser component. Figure 3 and Figure 4 show execution screens. The execution screens show a marked graph and its derivation tree.

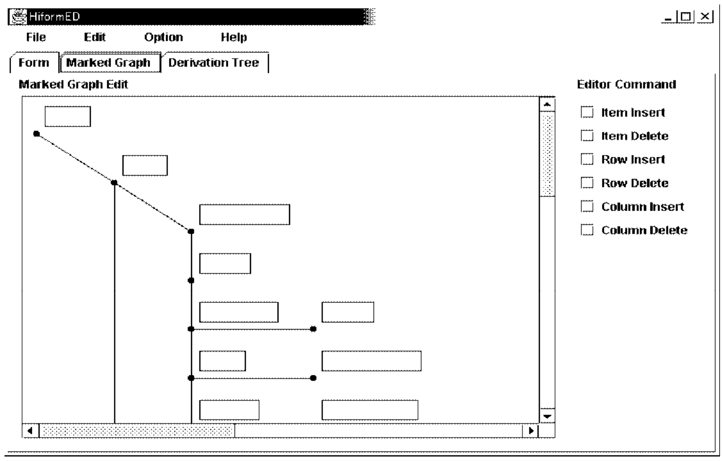

Figure 3: An Execution Screen of the Parsing Engine (Marked Graph)

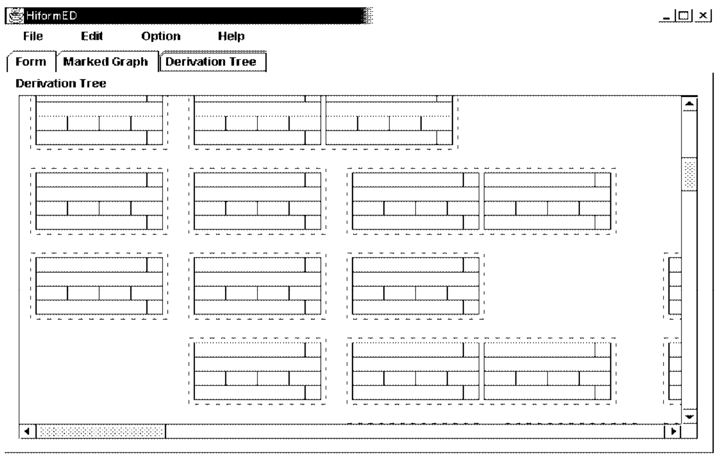

Figure 4: An Execution Screen of the Parsing Engine (Derivation Tree)

2.3 Classes for Marked Graphs and Derivation Trees

We developed a parsing engine by using Java language. We provide several class files for 2 models based on a graph grammar. Our model is a marked graph. The another model is a derivation tree. This parsing engine has several class files for constructing marked graphs and derivation trees. The class files for marked graphs are called MGC(Marked Graph Class) and the class files for derivation trees are called DTC(Derivation Tree Class).

A data of a marked graph is constructed by MGC. The parsing engine analyzes the data and makes a data of a derivation tree on DTC. Furthermore, the parsing engine evaluates attributes of the data of a derivation tree. A tabular form is drawn based on these attributes. The following Figure 5 illustrates the parsing process.

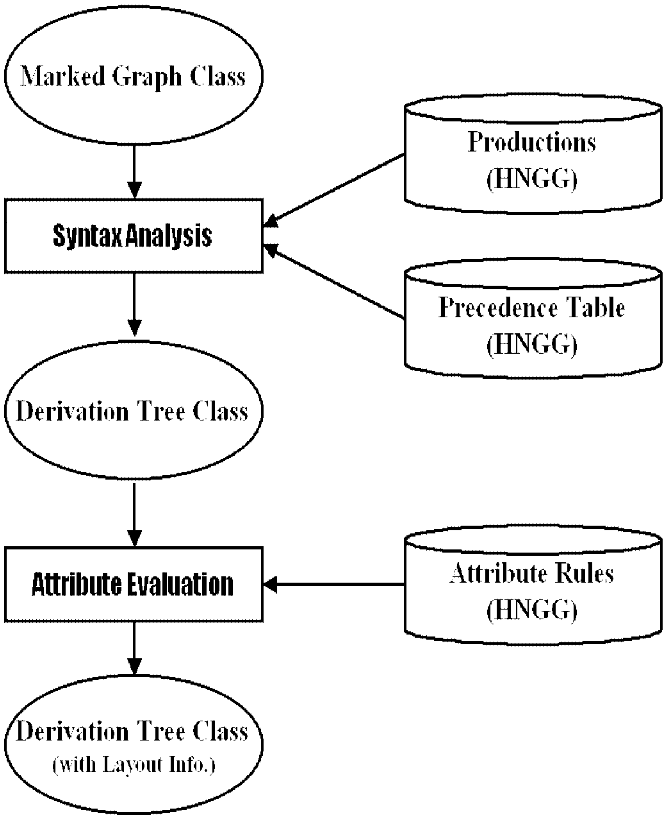

Figure 5: Parsing Process

Chapter 3

System Structure

Several file codes were proposed for diagrams. In 1997, H–Code2 [10] was proposed as a file code for structured–diagrams Hichart. H–Code2 is specidied based on BNF. In 1995, DXL [5] was proposed for exchanging diagram datas among diagram processing systems. Then DXL are applied to diagram representations such as NS–charts, Hichart, flow–charts etc.

Furthermore, several codes were developed for documents contained tables such as XML, HTML, MS–Excel files, TeX and so on. Some codes have hierartical structures. But these codes do not have graph–structures.

Since we specified tabular forms as graphs, our system needs a code that is able to represent graph strucuture naturally.

In this paper, we propose a tabular form code FXL based on graph structures.

3.1 System Overview [17, 19]

This system is constructed on a graph parsing engine for Hiform. This engine is constructed on three parts, which are productions for tabular form syntax, attribute rules for calculating values of tabular form's layout information and precedence table for tabular form parsing. We propose syntax-directed editing mechanism based on a graph grammar for tabular forms. Figure 6 illustrates the system overview.

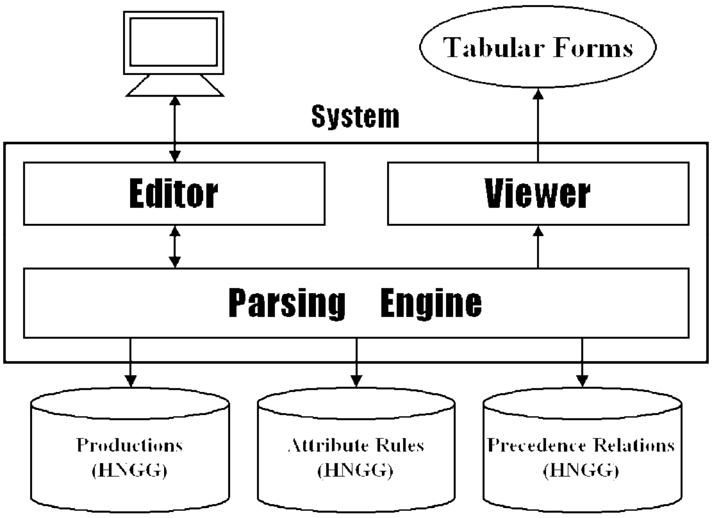

Figure 6: System Overview

3.2 File Structure [17, 19]

A File format is called FXL(a Form eXchange Language) which has graph structure. The FXL file is an external file of MGC. DTC is generated from MGC by syntax analysis and attribute evaluation. A viewer shows a tabular form by the DTC. Figure 7 illustrates a file structure.

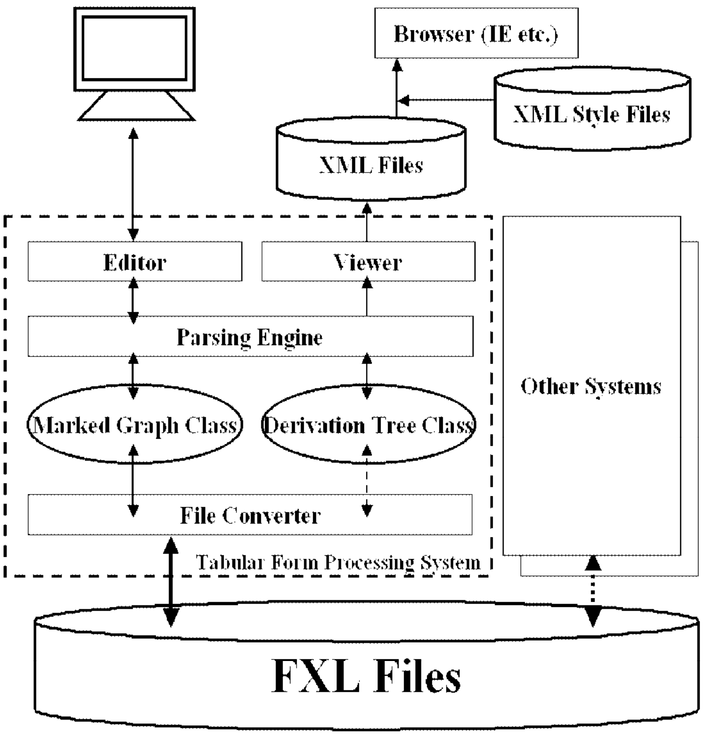

Figure 7: File Structure

Chapter 4

File Format

4.1 FXL [20]

FXL(A Form Exchange Language) is a file format of marked graph. FXL has following characteristics:

(1) Syntax of FXL is defined by extended BNF. Therefore, codes of FXL can be syntactically verified.

(2) Codes of FXL are text-based codes. Therefore, they can be edited directly.

(3) FXL can describe several attributes for tabular forms, which are locations and positions of cells and geometrical relations among cells.

Figure 8 and following source list show an example of FXL description. In Figure 8, the node 1 has a label "program name", the node 2 has a label "author", and the node 3 has a label "original release". Similarly, the edge (1, 2) has ID 1 and a label "ov", and the edge (2, 3) has ID 2 and a label "lf".

Detailed description of FXL is reported in Appendix.

An Example of FXL Description

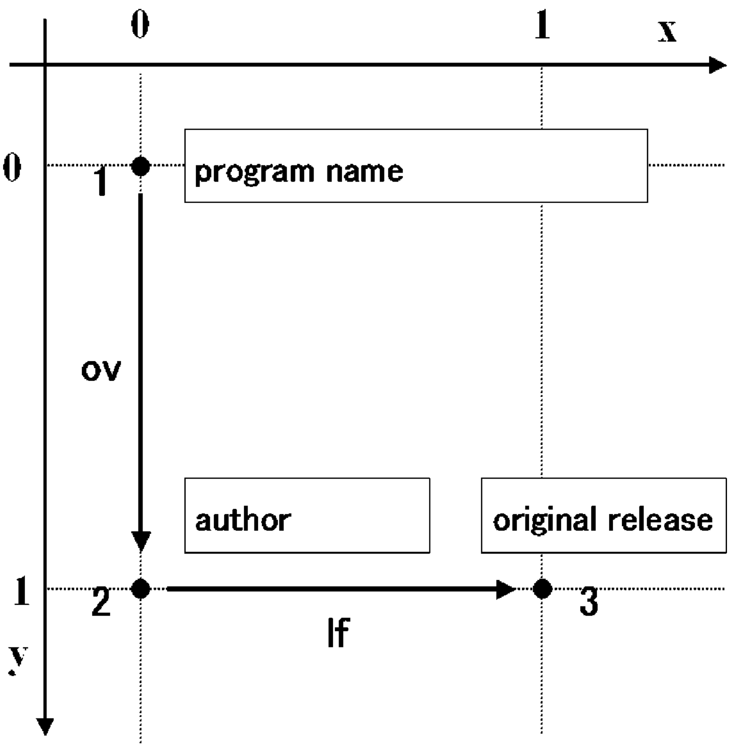

Figure 8: A Graph for Description Example of FXL

```
\\ FXL version0.01a

header{
    date{ 2000,11,24 }
    time{ 0,0,0 }
application{ "HiformED", "version 0.01a" }
}

graph{
    graphHeader{
    date{ 2002,1,1 }
    time{ 0,0,0 }
    }

  nodeSet{
```

```
nodeObject{
  node{
    nodeID{ 1 }
    nodeX{ 0 }
    nodeY{ 0 }
  }
  nodeLabel{
    labelString{ "program name" }
  }
  attribute{
    cellSize{
      cellWidth{ 2 }
      cellHeight{ 1 }
    }
    cellLocation{
      cellX{ 0 }
      cellY{ 0 }
    }
  }
  cellColor{
    fontRGB{ 0, 0, 0 }
  }
}
nodeObject{
  node{
    nodeID{ 2 }
    nodeX{ 0 }
    nodeY{ 1 }
  }
  nodeLabel{
    labelString{ "author" }
  }
  attribute{
    cellSize{
      cellWidth{ 1 }
      cellHeight{ 1 }
    }
    cellLocation{
      cellX{ 0 }
      cellY{ 1 }
    }
  }
  cellColor{
```

```
      fontRGB{ 0, 0, 0 }
    }
  }
  nodeObject{
    node{
      nodeID{ 3 }
      nodeX{ 1 }
      nodeY{ 1 }
    }
    nodeLabel{
      labelString{ "original release" }
    }
    attribute{
      cellSize{
        cellWidth{ 1 }
        cellHeight{ 1 }
      }
      cellLocation{
        cellX{ 1 }
        cellY{ 1 }
      }
    }
    cellColor{
      fontRGB{ 0, 0, 0 }
    }
}

edgeSet{
  edgeObject{
    edge{
      edgeID{ 1 }
      startNode{ 1 }
      endNode{ 2 }
      edgeShapes{ "arrow" }
    }
    edgeLabel{
      labelString{ "ov" }
    }
    edgeColor{
      fontCMYK{ 0, 0, 0, 100 }
    }
  }
  edgeObject{
```

```
      edge{
        edgeID{ 2 }
        startNode{ 2 }
        endNode{ 3 }
        edgeShapes{ "arrow" }
      }
      edgeLabel{
        labelString{ "lf" }
      }
      edgeColor{
        fontCMYK{ 0, 0, 0, 100 }
      }
    }
  }
}
```

FXL Source with the Structure of the Graph in Figure 8

4.2 Description of a Graph Part [20]

Structure of a Graph Part

```
graph{
    graphHeader {
        date{ ··· }
        time{ ··· }
    graphName{ graph_name }
}
        nodeSet{ nodeObject{ ··· }
        nodeObject{ ··· } ··· }
        edgeSet{ edgeObject{ ··· }
        edgeObject{ ··· } ··· }
}
```

The description of a graph part consists of 3 blocks, *graphHeader*, *nodeSet*, and *edgeSet*.

The part of "graphHeader" describes whole graph information. The part of "nodeSet" and "edgeSet" describe information of nodes and edges, respectively.

4.3 Description of a Node Part [20]

Structure of a Node Part

```
nodeObject{
    node{
        nodeID{ ID_Number }
        nodeX{ x }
        nodeY{ y }
    }
    nodeLabel{
        labelString{ label_string }
    }
    attribute{
        cellSize{
            cellWidth{ width }
            cellHeight{ height }
        }
        cellLocation{
            cellX{ x }
            cellY{ y }
        }
    }
    cellColor{
        fontRGB{ R, G, B }
    }
}
```

Information of a node is descrived in nodeObject part. The description of a node part consists of 2 blocks, *node* and *label.*

The part of "node" describes information on one node and information which accompanies its node. The part of *nodeID*{ *ID_Number* } describes the discernment information on the node. The part of *nodeX*{ x } and the part of *nodeY*{ y } describe x-coordinate and y-coordinates of the node, respectively.

The part of "nodeLabel" describes information on a label which accompanies the node. The *labelString*{ *label_string* } describes a string for a label name.

The part of "attribute" describes information on attributes which accompanies the node. The *cellSize* describes width and height of a cell in a tabular form. And the *cellLocation* describes coordinates of the cell in a tabular form.

4.4 Description of an Edge Part [20]

Structure of an Edge Part

```
edgeObject{
    edge{
        edgeID{ { edge_ID } }
        startNode{ { start_node } }
        endNode{ { end_node } }
        edgeShape{ { shape } }
    }
    edgeLabel{
        labelString{ label_string }
    }
    edgeColor{
        fontCMYK{ C, M, Y, K }
    }
}
```

Information of an edge is descrived in edgeObject part. The description of an edge part consists of 2 blocks, *edge* and *label.*

The part of "edge" describes information on one edge and information which accompanies its edge. The part of *edgeID*{ *edge_ID* } describes the discernment information on the edge. The part of *startNode*{ *start_node* } and the part of *endNode*{ *end_node* } describe an ID number of the start node and an ID number of the terminal node, respectively.

The part of "label" describes information on a label which accompanies the edge. The *labelString*{ *label_string* } describes a string for a label name.

Chapter 5

Conclusion

We dealt with syntactic tabular form designing environment, based on attribute NCE graph grammars. We proposed the system structure and the file structures of the environment.

We constructed a universal format FXL for a tabular form processing system. A file described by FXL is able to represent the locations and sizes of items in a tabular form. This file format could be applied to other tabular form processing systems.

In the future, we are planning the development of parsing engine, editor, and viewer based on this file format FXL.

References

[1] Reinhold Franck, A Class of Linearly Parsable Graph Grammars, Acta Infomatica 10, 175-201 (1978)

[2] ISO6592-1985, Guidelines for the Documentation of Computer-Based Application Systems (1985).

[3] A.V. エイホ, R. セシィ, J.D. ウルマン共著, 原田賢一訳, コンパイラ　原理・技法・ツール, サイエンス社 (1990)

[4] 小野木一樹, H コード言語解説書 ver. 5.0, Hichart 研究会資料 91-04 (1991)

[5] Information technology–DXL : Diagram exchange language for tree–structured charts, JIS X 0130 (1995).

[6] Y. Adachi, K. Anzai et al., Hierarchical Program Diagram Editor Based on Attribute Graph Grammar, *Proc. COMPSAC96*, 205-213 (1996).

[7] Grsegorz Rozenberg (Ed.), Handbook of Graph Grammar and Computing by Graph Transformation, World Scientific Publishing (1997).

[8] G. Santucci and L. Tarantino, A Hypertabular Visualizer of Query Results, *Proc. of the 1997 IEEE Symp. Visual Language*, 193-200 (1997).

[9] K. Sugita, Y. Adachi, Y. Miyadera, K. Tsuchida and T. Yaku, Advanced Software Mechanisms for Computer-Aided Instruction Information Literacy, *APEC-CIL*'97 (1997).

[10] Youzou Miyadera, Takeo Yaku, Hideaki konya, An Expression Method for Circulation od Tree–Structured Diagrams, 東京電機大学理工学部紀要 vol.19 No.1, 41-59 (1997)

[11] K. Sugita, A. Adachi, Y. Miyadera, K. Tsuchida and T. Yaku, A Visual Programming Environment Based on Graph Grammars and Tidy Graph Drawing, *Proc. Internat. Conf. Software Engin.* (ICSE '98) 20-II, 74-79 (1998).

[12] A. Adachi, T. Tsuchida and T. Yaku, Program Visualization Using Attribute Graph Grammars, CD-ROM Book, *IFIP World Computer Congress* 98 (1998).

[13] T. Arita, K. Tomiyama, T. Yaku, Y. Miyadera, K. Sugita, K. Tsuchida, Syntactic Processing of Diagrams by Graph Grammars, *Proc. IFIP WCC ICS 2000*,145-151 (2000).

[14] 有田友和, グラフに対するデータ構造記述フォーマット, プライベートメモ (2000).

[15] O. Koeth and M. Minas, Abstraction in Graph–Transformation Based Diagram Editors, Electric Notes in Theoretical Computer Science vol.50, 3 (2001).

[16] R.Bardohls, T. Schultzke and G. Taentzer, Visual Language Parsing in GenGEd, Electric Notes in Theoretical Computer Science, vol 50, 3 (2001).

[17] T. Arita, S. Nakagawa, K. Tomiyama, K. Tsuchida, T. Yaku, A Syntax Directed Environment of Tabular Form Designing, *ICSE 2001* (2001).

[18] T. Arita, K. Sugita, K. Tsuchida, T. Yaku, Syntactic Tabular Form Processing by Precedence Attribute Graph Grammars, *Proc. IASTED Applied Informatics 2001*, 637-642 (2001).

[19] O. Inoue, K. Tsuchida, S. Nakagawa, T. Arita, T. Yaku, An XML Viewer for Tabular Forms for use with mechanical Documentation, *IASTED AI'02* (2002). (to appear)

[20] T. Arita, S. Nakagawa, K. Tsuchida, T. Yaku, A Universal File Structure for Tabular Form Processing Systems, 第5回プログラミングおよび応用のシステムに関するワークショップ SPA '02 (2002). (to appear)

[21] Extensible Markup Language (XML), The World Wide Web Consortium (W3C), http://www.w3.org/XML.

List of Papers by Shun-ichi NAKAGAWA

1. † A Syntax Directed Environment of Tabular Form Designing
   T. Arita, S. Nakagawa, K. Tomiyama, K. Tsuchida, and T. Yaku
   ICSE 2001 (2001).

2. ‡ A Syntax Directed Environment of Tabular Form Processing
   T. Arita, S. Nakagawa, K. Tomiyama, Y. Miyadera, K. Tsuchida, and T. Yaku
   電子情報通信学会 2001 年総合大会講演論文集 (D-3-7), 35 (2001)

3. ‡ An XML Viewer for Tabular Forms for use with mechanical Documentation
   O. Inoue, K. Tsuchida, S. Nakagawa, T. Arita, and T. Yaku
   IASTED AI'02 (2002). (to appear)

4. ‡ An XML Viewer for Tabular Forms in Mechanical Documentation
   O. Inoue, S. Nakagawa, T. Arita, T. Yaku, and K. Tsuchida
   電子情報通信学会 ソフトウェアサイエンス研究会, 信学技報, Vol.101, No.629, 31-38 (2002).

5. ‡ FXL : A Form Exchange Language of Modular Forms for Program Specification Documents
   T. Arita, S. Nakagawa, K. Tsuchida, and T. Yaku
   第 5 回プログラミングおよび応用のシステムに関するワークショップ SPA '02 (2002). (to appear)

† 主論文, ‡ 参考論文