FXL : A Form Exchange Language of Modular Forms for Program Specification Documents

Tomokazu ARITA[†] Shun-ichi NAKAGAWA[†] Kensei TSUCHIDA[††] and Takeo YAKU[†]

[†]Dept. Comput. Sci. & System Analysis, Nihon Univ.
Setagaya, 156-8550, Japan
{arita, nakagawa, yaku}@cssa.chs.nihon-u.ac.jp

[††]Dept. Inform. & Comput. Engin., Toyo Univ.
Kawagoe, 350-8585, Japan
kensei@eng.toyo.ac.jp

Abstruct 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.

keywords software documentation, visualization, graph grammars, tabular forms

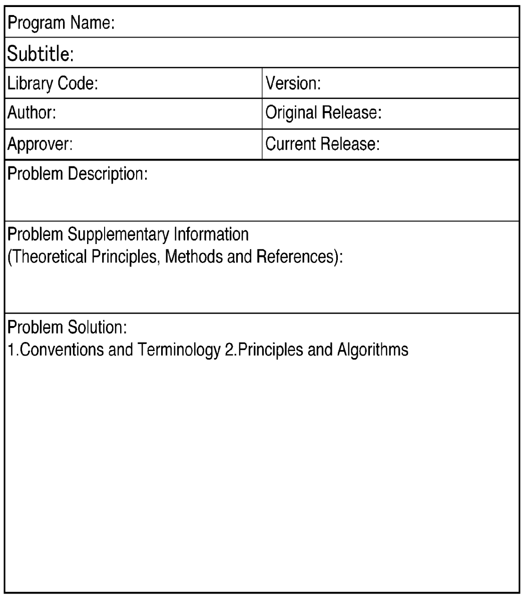
1 Introduction

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.

In 1985, ISO6592 was issued as a guideline for program documentation. There is all items for program specification documents in ISO6592. However, ISO6592 does not include descriptions for document styles. So, Hiform was proposed as tabular forms for program specification documents based on ISO6592. As the characteristic of tabular forms, tabular forms can include various description styles that are letters and diagrams such as UML, Flowcharts etc. This paper deals with a tabular form processing system. 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. We formulated the hierarchical structured diagram [4, 7].

In the 1980s, Hichart, PAD, SPD, and HCP were proposed as research of program diagrams. And H-code2 of list form was proposed as internal code for program diagrams. In 1995, DXL[3] was proposed as a universal code of them and defined by BNF. Recently, XML[11] is proposed in order to a universal format for structured documents. Furthermore several types of formats based on XML is also proposed such as XMI for UML, GXL for Graph and so on.

In 2000, we introduced partly a syntactic definition of program specification forms based on ISO6592 standard [8, 9]. We employed graph grammars for formalizing those forms in [8]. In this paper, we deals with a universal processing system for tabular forms. We propose system overview, and inner codes for marked graphs (called MGC). Furthermore we also propose a data format FXL for tabular forms based on MGC. This data format could be applied other modular tabular forms such as symbol tables, specification forms etc. This



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

paper is organized as follows.

In Section 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 Section 3, we introduce a syntactic processing system and the file structures using a parser for tabular forms, which provides mechanical verifier and drawer. In Section 4, we propose a file format for our tabular form processing system. In Section 5, we summarize our results.

2 Preliminaries

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.

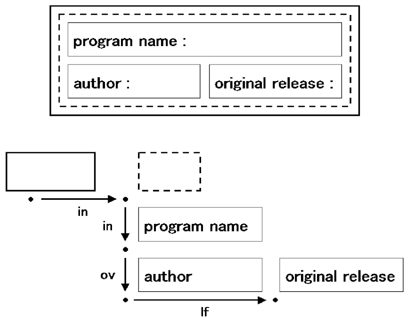


Figure 2: Nested tabular form and its corresponding marked graph

2.1 Tabular Forms for Program Specification

We consider here a program specification language called *Hiform* [5] 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. We considered the ISO6592 items and introduced Hiform, which includes all items defined in these Annexes. Hiform [6] 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. [5]).

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 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'.

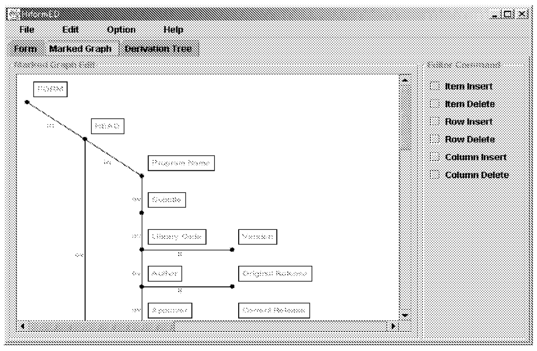


Figure 3: An Execution Screen of the Parsing Engine (Marked Graph)

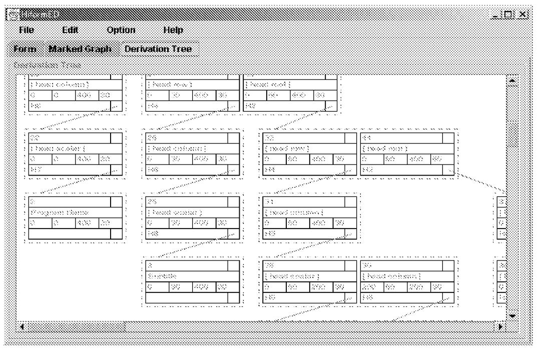


Figure 4: An Execution Screen of the Parsing Engine (Derivation Tree)

2.2 HNGG [9][10]

Hiform documents is formalized based on a graph grammar, which called HNGG. HNGG has some characteristics as following: (1) HNGG's productions is syntax of tabular forms. Productions are used for deciding relations between items and structure of tabular forms. (2) HNGG is a precedence graph grammar. HNGG has precedence relations for efficient parsing. A precedence relation can be used for supporting a search of a handle on syntactic analysis. (3) HNGG has attribute rules. Attribute rules are used for calculating values for layout information of tabular forms and XML source. There are attributes for layout information such as locations and sizes of items

These characteristics show that HNGG provides the structure of tabular forms and drawing conditoin of them. Furthermore, editing manipulations for tabular forms are proposed based on HNGG [10].

2.3 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.

The Figure 5 illustrates the parsing process. First, by syntax analysis for a marked graph with attribute, a derivation tree is generated. Our parsing engine refers to not only productions but also precedence tables. The marked graph is analyzed efficiently by using precedence tables. 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 viewer component. Figure 3 and Figure 4 show execution screens. The execution screens show a marked graph and its derivation tree.

3 System Structure

3.1 System Overview

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.

3.2 File Structure

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. This system has a four components for data processing. Editor components provides editing manipulation for tabular

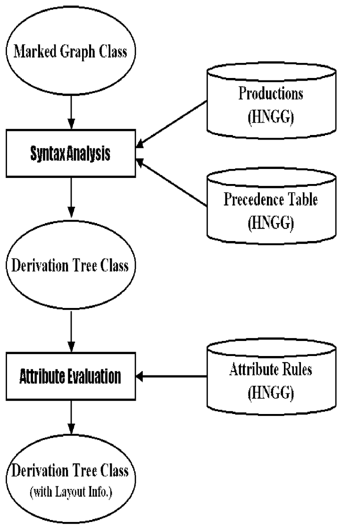


Figure 5: Parsing Process

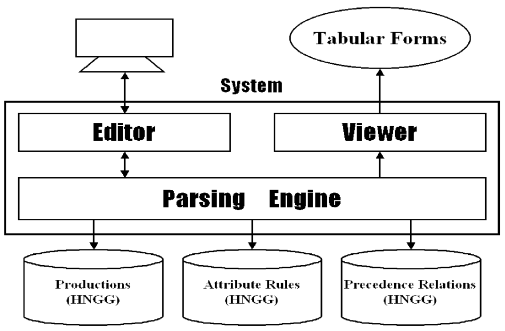


Figure 6: System Overview

form. This components edits marked graphs directly. Parsing Engine analyzes marked graphs and generated derivation trees with attributes. Viewer components generates XML files from derivation trees for other systems. This XML file displays other XML browsers by using XML style sheets based on XSL. File converter generates FXL files form marked graphs. Editor components and File converter are not developed yet.

3.3 MGC and DTC

We developed a parsing engine by using Java language. We provide several class files for 2 models

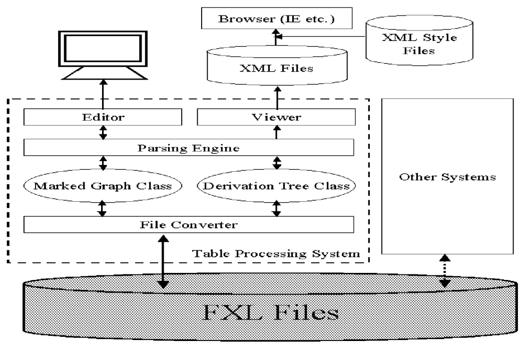


Figure 7: File Structure

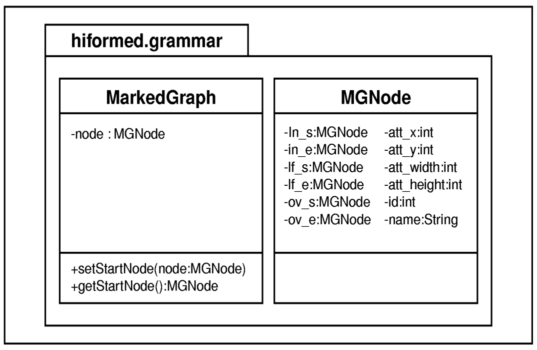


Figure 8: MGC: classes for the marked graph

based on a graph grammar. One model is a marked graph. The other 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.

3.4 The Data Structure of MGC

This parsing engine is implemented by Java Language. Of course, the marked graph and the derivation tree is also implemented by Java Language. Our package of Java provides Java classes for mod-

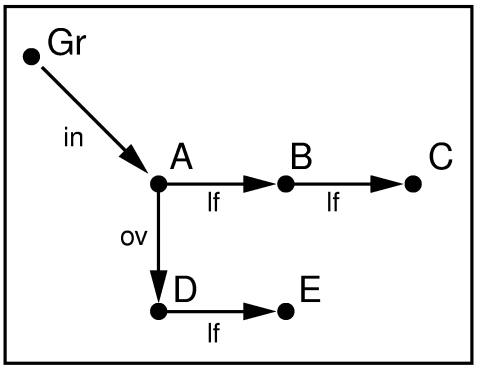


Figure 9: a marked graph G

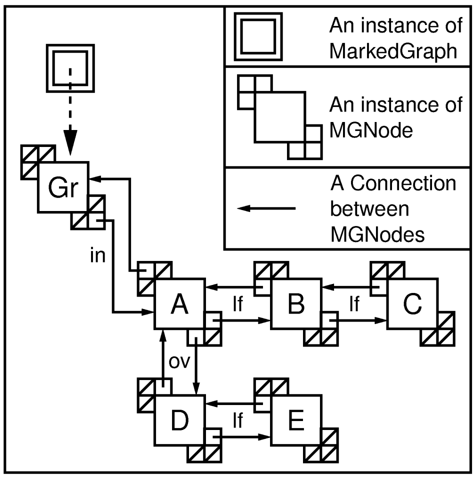


Figure 10: A data structure by MGC for G

eling our graph grammar.

The marked graph is implemented by using MarkedGraph.class and MGNode.class (we called MGC). Figure 8 illustrates classes for modeling marked graphs. The marked graph is represented by a list in our package. Figure 10 illustrates an instance of MarkedGraph.class for a marked graph in Figure 9.

4 Data Format

4.1 FXL

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

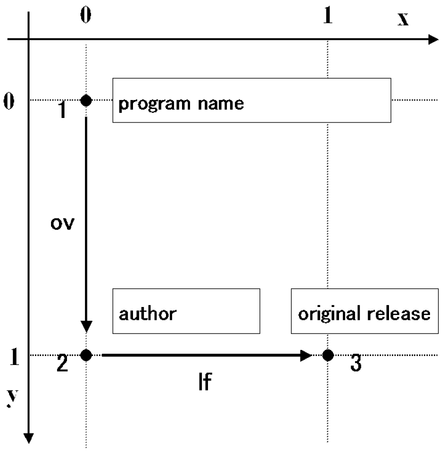


Figure 11: A Graph for Description Example of FXL

(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 11 is a graph representation for FXL. Appendix 1 shows an example of FXL description for Figure 11. In Figure 11, 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".

4.1.1 Description of an Graph Part

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.

4.1.2 Description of a Node Part

Structure of a Node Part

```
nodeObject{
    node{
        nodeID{ ID_Number }
        nodeX{ x }
        nodeY{ y }
    }
    label{
        labelString{ label_string }
        cellSize{
            cellWidth{ cell_width_inner }
            cellHeight{ cell_height_inner }
        }
        cellLocation{
            cellX{ cell_x_inner }
            cellY{ cell_y_inner }
        }
    }
}
```

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

4.1.3 Description of an Edge Part

Structure of an Edge Part

```
edgeObject{
    edge{
        edgeID{ { edge_ID } }
        startNode{ { start_node } }
        endNode{ { end_node } }
        edgeShape{ { shape } }
    }
    label{
        labelString{ label_string }
    }
}
```

Information of an edge is described 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.

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 data format could be applied to other tabular form processing systems and to modular tabular forms.

The size of graphs described by MGC is larger than the size of graphs generated by HNGG. Furthermore, The size of graphs described by FXL is larger than MGC. In the future, the size of FXL needs to correspond to the size of HNGG.

The development of parsing engine, editor, and viewer based on this data format FXL is still remained.

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

[4] Y. Adachi, K. Anzai et al., Hierarchical Program Diagram Editor Based on Attribute Graph Grammar, *Proc. COMPSAC*96, 205-213(1996).

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

[6] 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).

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

[8] 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).

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

[10] T. Arita, K. Tomiyama, K. Tsuchida and T. Yaku, Application of Attribute NCE Graph Grammars to Syntactic Editing of Tabular Forms, Electric Notes in Theoretical Computer Science, Vol. 50, 3, (2001).

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

Appendix 1

```
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 }
      }
      label{
        labelString{ “ program name ” }
        cellSize{
          cellWidth{ 2 }
          cellHeight{ 1 }
        cellLocation{
          cellX{ 0 }
          cellY{ 0 }
      }
    }
    nodeObject{
      node{
        nodeID{ 2 }
        nodeX{ 0 }
        nodeY{ 1 }
      }
      label{
        labelString{ “ author ” }
        cellSize{
          cellWidth{ 1 }
          cellHeight{ 1 }
        cellLocation{
          cellX{ 0 }
          cellY{ 1 }
      }
    }
    nodeObject{
      node{
        nodeID{ 3 }
        nodeX{ 1 }
        nodeY{ 1 }
      }
      label{
        labelString{ “ original release ” }
        cellSize{
          cellWidth{ 1 }
          cellHeight{ 1 }
        cellLocation{
          cellX{ 1 }
          cellY{ 1 }
      }
    }
  }

  edgeSet{
    edgeObject{
      edge{
        edgeID{ 1 }
        startNode{ 1 }
        endNode{ 2 }
        edgeShapes{‘‘arrow’’}
      }
      label{
        labelString{ “ ov ” }
      }
    }
    edgeObject{
      edge{
        edgeID{ 2 }
        startNode{ 2 }
        endNode{ 3 }
        edgeShapes{‘‘arrow’’}
      }
      label{
        labelString{ “ lf ” }
      }
    }
  }
}
```

An Example of description of FXL